

3D Viewer

Generated by Doxygen 1.9.1

1 s21_3DViewer	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 Data Struct Reference	7
4.1.1 Detailed Description	7
4.1.2 Member Data Documentation	8
4.1.2.1 facets_count	8
4.1.2.2 gif_start	8
4.1.2.3 polygons	8
4.1.2.4 scale	8
4.1.2.5 vertex_count	8
4.1.2.6 vertexes	8
4.2 facets Struct Reference	9
4.2.1 Detailed Description	9
4.2.2 Member Data Documentation	9
4.2.2.1 count_v	9
4.2.2.2 plnds	9
4.3 matrix_struct Struct Reference	9
4.3.1 Detailed Description	10
4.3.2 Member Data Documentation	10
4.3.2.1 columns	10
4.3.2.2 matrix	10
4.3.2.3 rows	10
5 File Documentation	11
5.1 parser/parser.c File Reference	11
5.1.1 Detailed Description	11
5.1.2 Function Documentation	12
5.1.2.1 memory_handling()	12
5.1.2.2 parser()	12
5.2 parser/parser.h File Reference	13
5.2.1 Detailed Description	15
5.2.2 Typedef Documentation	15
5.2.2.1 data	15
5.2.2.2 matrix_t	15
5.2.2.3 polygon_t	15
5.2.3 Enumeration Type Documentation	15
5.2.3.1 Rotation	15

5.2.4 Function Documentation	16
5.2.4.1 check_mx_allocation()	16
5.2.4.2 check_row_allocation()	16
5.2.4.3 check_symbol()	17
5.2.4.4 create_matrix()	17
5.2.4.5 error_free()	17
5.2.4.6 free_data()	18
5.2.4.7 is_facet()	18
5.2.4.8 is_vertex()	18
5.2.4.9 memory_handling()	19
5.2.4.10 parser()	19
5.2.4.11 remove_matrix()	20
5.2.4.12 setNewScale()	20
5.2.4.13 setScaling()	20
5.3 parser/parser_aux.c File Reference	21
5.3.1 Detailed Description	22
5.3.2 Function Documentation	22
5.3.2.1 check_mx_allocation()	22
5.3.2.2 check_row_allocation()	22
5.3.2.3 check_symbol()	23
5.3.2.4 create_matrix()	23
5.3.2.5 error_free()	24
5.3.2.6 free_data()	24
5.3.2.7 is_facet()	24
5.3.2.8 is_vertex()	25
5.3.2.9 remove_matrix()	25
5.3.2.10 setNewScale()	25
5.3.2.11 setScaling()	26
5.4 transform/transform.c File Reference	26
5.4.1 Detailed Description	27
5.4.2 Function Documentation	27
5.4.2.1 affineMovingOperation()	27
5.4.2.2 affineRotationOperation()	28
5.4.2.3 affineScalingOperation()	29
5.4.2.4 getSign()	29
5.4.2.5 rotation_by_ox()	29
5.4.2.6 rotation_by_oy()	30
5.4.2.7 rotation_by_oz()	30
5.4.2.8 scaling()	30
5.4.2.9 setFigureToCenter()	31
5.5 transform/transform.h File Reference	31
5.5.1 Detailed Description	33

5.5.2 Enumeration Type Documentation	33
5.5.2.1 MOVING_TO_POSITION	33
5.5.2.2 Scaling	33
5.5.3 Function Documentation	34
5.5.3.1 affineMovingOperation()	34
5.5.3.2 affineRotationOperation()	34
5.5.3.3 affineScalingOperation()	35
5.5.3.4 getSign()	35
5.5.3.5 rotation_by_ox()	36
5.5.3.6 rotation_by_oy()	36
5.5.3.7 rotation_by_oz()	36
5.5.3.8 scaling()	37
5.5.3.9 setFigureToCenter()	37
Index	39

Chapter 1

s21_3DViewer

This program for viewing 3D wireframe models (3D Viewer) in the C programming language. The models themselves must be loaded from .obj files and be viewable on screen with the ability to rotate, scale and translate. A wireframe model is a model of an object in 3D graphics, which is a set of vertices and edges that defines the shape of the displayed polyhedral object in three-dimensional space.

The program provides the ability to:

- Load a wireframe model from an obj file (vertices and surfaces list support only);
- Translate the model by a given distance in relation to the X, Y, Z axes;
- Rotate the model by a given angle relative to its X, Y, Z axes;
- Scale the model by a given value;

The program allows customizing the type of projection (parallel and central);

The program allows setting up the type (solid, dashed), color and thickness of the edges, display method (none, circle, square), color and size of the vertices;

The program allows choosing the background color;

Settings can be saved between program restarts.

The program allows saving the captured (rendered) images as bmp and jpeg files;

The program allows recording small screencasts by a special button - the current custom affine transformation of the loaded object into gif-animation.

The graphical user interface contains:

- A button to select the model file and a field to output its name;
- A visualisation area for the wireframe model;
- Buttons and input fields for translating the model;
- Buttons and input fields for rotating the model;
- Buttons and input fields for scaling the model;
- Information about the uploaded model - file name, number of vertices and edges;

Use standard set of Makefile targets: all, install, uninstall, clean, dvi, dist, tests, gcov.

Don't forget to specify your own qmake and installation path in Makefile.

Chapter 2

Class Index

2.1 Class List

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

Data	A structure representing the data to be displayed	7
facets	A structure representing the edges of the model	9
matrix_struct	A structure representing the matrix	9

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

parser/ parser.c	
File to process the file and get vertices and polygons	11
parser/ parser.h	
Header file for file processing	13
parser/ parser_aux.c	
Auxiliary file for file processing	21
transform/ transform.c	
Affine operations implementation file	26
transform/ transform.h	
Header file of affine operations	31

Chapter 4

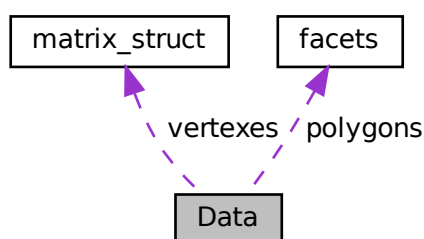
Class Documentation

4.1 Data Struct Reference

A structure representing the data to be displayed.

```
#include <parser.h>
```

Collaboration diagram for Data:



Public Attributes

- unsigned int [vertex_count](#)
- unsigned int [facets_count](#)
- [polygon_t](#) * [polygons](#)
- [matrix_t](#) [vertexes](#)
- double [scale](#) [6]
- int [gif_start](#)

4.1.1 Detailed Description

A structure representing the data to be displayed.

This structure contains information about the number of vertices and faces, an array of faces, a matrix of vertices, scaling and start flag for GIF animation.

4.1.2 Member Data Documentation

4.1.2.1 facets_count

```
unsigned int Data::facets_count
```

Number of faces.

4.1.2.2 gif_start

```
int Data::gif_start
```

Flag for the start of GIF animation.

4.1.2.3 polygons

```
polygon_t* Data::polygons
```

Array of faces.

4.1.2.4 scale

```
double Data::scale[6]
```

Scaling array.

4.1.2.5 vertex_count

```
unsigned int Data::vertex_count
```

Number of vertices.

4.1.2.6 vertexes

```
matrix_t Data::vertexes
```

Matrix of vertices.

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

- [parser/parser.h](#)

4.2 facets Struct Reference

A structure representing the edges of the model.

```
#include <parser.h>
```

Public Attributes

- unsigned int [count_v](#)
- unsigned int * [pInds](#)

4.2.1 Detailed Description

A structure representing the edges of the model.

This structure contains information about the faces of the model, including the number of vertices and the indices of the vertices that form the face.

4.2.2 Member Data Documentation

4.2.2.1 count_v

```
unsigned int facets::count_v
```

Number of vertices forming a face.

4.2.2.2 pInds

```
unsigned int* facets::pInds
```

Array of indices of the vertices that form the face.

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

- [parser/parser.h](#)

4.3 matrix_struct Struct Reference

A structure representing the matrix.

```
#include <parser.h>
```

Public Attributes

- double ** [matrix](#)
- int [rows](#)
- int [columns](#)

4.3.1 Detailed Description

A structure representing the matrix.

This structure contains a two-dimensional array of matrix elements, as well as the number of rows and columns of the matrix.

4.3.2 Member Data Documentation

4.3.2.1 columns

```
int matrix_struct::columns
```

Number of matrix columns.

4.3.2.2 matrix

```
double** matrix_struct::matrix
```

Two-dimensional array of matrix elements.

4.3.2.3 rows

```
int matrix_struct::rows
```

Number of matrix rows.

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

- [parser/parser.h](#)

Chapter 5

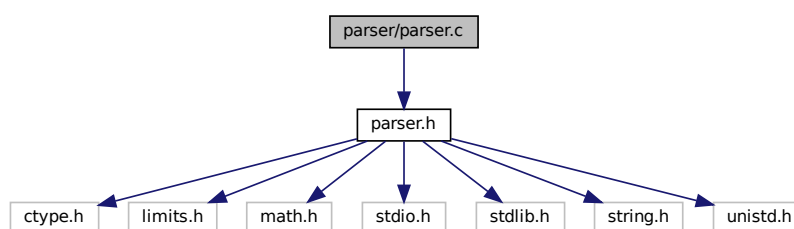
File Documentation

5.1 parser/parser.c File Reference

file to process the file and get vertices and polygons

```
#include "parser.h"
```

Include dependency graph for parser.c:



Functions

- int `memory_handling` (data *drawing_data)
Function for allocating memory for polygons.
- int `parser` (data *drawing_data, char *filename)
Parses data from a file and populates the data structure.

5.1.1 Detailed Description

file to process the file and get vertices and polygons

Author

mitchelk, nenamaxi and dannamer

Version

0.1

Date

2024-03-22

Copyright

Copyright (c) 2024

5.1.2 Function Documentation

5.1.2.1 memory_handling()

```
int memory_handling (
    data * drawing_data )
```

Function for allocating memory for polygons.

Parameters

<i>drawing_data</i>	A <code>data</code> type structure for storing information about vertices.
---------------------	--

Returns

Returns the error code(PARSER_OK or PARSER_FALSE).

5.1.2.2 parser()

```
int parser (
    data * drawing_data,
    char * filename )
```

Parses data from a file and populates the data structure.

This method parses data from the specified file and populates the appropriate data structure information about the vertices and faces of the model.

Parameters

<i>drawing_data</i>	Pointer to the data structure into which the parsing result will be written.
<i>filename</i>	The name of the file to parse.

Returns

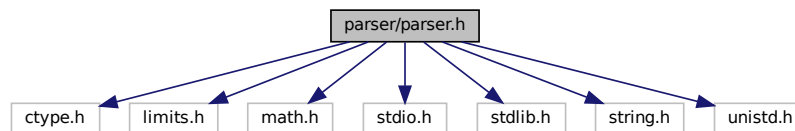
int Parsing error code (PARSER_OK if successful).

5.2 parser/parser.h File Reference

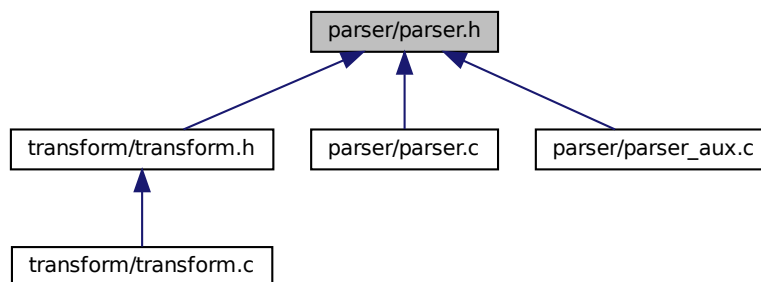
Header file for file processing.

```
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

Include dependency graph for parser.h:



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



Classes

- struct [facets](#)
A structure representing the edges of the model.
- struct [matrix_struct](#)
A structure representing the matrix.
- struct [Data](#)
A structure representing the data to be displayed.

Macros

- `#define ST_SIZE_COORDINATE 3`
Macro for coordinate size.
- `#define PARSER_OK 0`
Macro for parser error code.
- `#define PARSER_FALSE 1`
Macro for parser error code.

Typedefs

- `typedef struct facets polygon_t`
A structure representing the edges of the model.
- `typedef struct matrix_struct matrix_t`
A structure representing the matrix.
- `typedef struct Data data`
A structure representing the data to be displayed.

Enumerations

- `enum Rotation {`
 `XM = 0 , XP = 1 , YM = 2 , YP = 3 ,`
 `ZM = 4 , ZP = 5 }`
Enumeration structure for affine rotation operation.
- `enum INDEX_COORDINATE { X = 0 , Y = 1 , Z = 2 }`
Enumeration structure for coordinade.

Functions

- `int parser (data *drawing_data, char *filename)`
Parses data from a file and populates the data structure.
- `int memory_handling (data *drawing_data)`
Function for allocating memory for polygons.
- `int create_matrix (int rows, int columns, matrix_t *result)`
Creates a matrix of the given dimensions.
- `void remove_matrix (matrix_t *A)`
Frees the memory allocated for the matrix.
- `void check_mx_allocation (double **mx, int *error)`
*Checks if memory for matrix double *array was actually allocated.*
- `void check_row_allocation (matrix_t *mx, int index, int *error)`
Checks if memory for matrix double array was actually allocated.
- `void error_free (matrix_t *mx, int index)`
Frees the memory allocated for the matrix.
- `int check_symbol (const char ch, const char compCh)`
Checks whether the character ch is equal to the character compCh.
- `int is_vertex (char *buffer)`
if current string contains vertex information
- `int is_facet (char *buffer)`
if current string contains facet information
- `void free_data (data *drawing_data)`
Frees memory occupied by the model data structure.
- `void setNewScale (data *drawing_data)`
Sets a new model scale based on extreme values in the X, Y, and Z axes.
- `void setScaling (const double minValue, const double maxValue, data *data_)`
Sets the scale to display the model.

5.2.1 Detailed Description

Header file for file processing.

Author

mitchelk, nenamaxi and dannamer

Version

0.1

Date

2024-03-22

Copyright

Copyright (c) 2024

5.2.2 Typedef Documentation

5.2.2.1 data

```
typedef struct Data data
```

A structure representing the data to be displayed.

This structure contains information about the number of vertices and faces, an array of faces, a matrix of vertices, scaling and start flag for GIF animation.

5.2.2.2 matrix_t

```
typedef struct matrix_struct matrix_t
```

A structure representing the matrix.

This structure contains a two-dimensional array of matrix elements, as well as the number of rows and columns of the matrix.

5.2.2.3 polygon_t

```
typedef struct facets polygon_t
```

A structure representing the edges of the model.

This structure contains information about the faces of the model, including the number of vertices and the indices of the vertices that form the face.

5.2.3 Enumeration Type Documentation

5.2.3.1 Rotation

```
enum Rotation
```

Enumeration structure for affine rotation operation.

Enumerator

XM	Field indicating the direction to the left
XP	Field indicating the direction to the right
YM	Field indicating the direction to the down
YP	Field indicating the direction to the up
ZM	Near
ZP	Far

5.2.4 Function Documentation

5.2.4.1 check_mx_allocation()

```
void check_mx_allocation (
    double ** mx,
    int * error )
```

Checks if memory for matrix double *array was actually allocated.

Parameters

<i>mx</i>	a pointer to a array of double * variables
<i>error</i>	a pointer to an error variable

5.2.4.2 check_row_allocation()

```
void check_row_allocation (
    matrix_t * mx,
    int index,
    int * error )
```

Checks if memory for matrix double array was actually allocated.

Parameters

<i>mx</i>	a matrix_t pointer
<i>index</i>	a number of allocated array elements
<i>error</i>	a pointer to an error code

5.2.4.3 check_symbol()

```
int check_symbol (
    const char ch,
    const char compCh )
```

Checks whether the character *ch* is equal to the character *compCh*.

This function compares the character *ch* with the character *compCh* and returns 1 if they are equal, or 0 otherwise.

Parameters

<i>ch</i>	The character to check.
<i>compCh</i>	The character to compare <i>ch</i> with.

Returns

Returns 1 if the characters are equal, 0 otherwise.

5.2.4.4 create_matrix()

```
int create_matrix (
    int rows,
    int columns,
    matrix_t * result )
```

Creates a matrix of the given dimensions.

This method creates a matrix with the specified number of rows and columns and initializes all its elements to zero.

Parameters

<i>rows</i>	Number of rows in the matrix.
<i>columns</i>	The number of columns in the matrix.
<i>result</i>	Pointer to the data structure into which the result of creating the matrix will be written.

Returns

int Matrix creation error code (PARSER_OK if successful).

5.2.4.5 error_free()

```
void error_free (
    matrix_t * mx,
    int index )
```

Frees the memory allocated for the matrix.

This function frees memory allocated for a matrix of type `matrix_t`.

Parameters

<i>mx</i>	Pointer to a <code>matrix_t</code> structure containing the matrix.
<i>index</i>	Index indicating the number of rows of the matrix.

5.2.4.6 free_data()

```
void free_data (
    data * drawing_data )
```

Frees memory occupied by the model data structure.

This method frees the memory allocated for storing the vertices and faces of the model.

Parameters

<i>drawing_data</i>	Pointer to the model data structure.
---------------------	--------------------------------------

5.2.4.7 is_facet()

```
int is_facet (
    char * buffer )
```

if current string contains facet information

Parameters

<i>buffer</i>	
---------------	--

Returns

1 -yes, 2 - no

5.2.4.8 is_vertex()

```
int is_vertex (
    char * buffer )
```

if current string contains vertex information

Parameters

<i>buffer</i>	
---------------	--

Returns

1 -yes, 2 - no

5.2.4.9 memory_handling()

```
int memory_handling (
    data * drawing_data )
```

Function for allocating memory for polygons.

Parameters

<i>drawing_data</i>	A data type structure for storing information about vertices.
---------------------	---

Returns

Returns the error code(PARSER_OK or PARSER_FALSE).

5.2.4.10 parser()

```
int parser (
    data * drawing_data,
    char * filename )
```

Parses data from a file and populates the data structure.

This method parses data from the specified file and populates the appropriate data structure information about the vertices and faces of the model.

Parameters

<i>drawing_data</i>	Pointer to the data structure into which the parsing result will be written.
<i>filename</i>	The name of the file to parse.

Returns

int Parsing error code (PARSER_OK if successful).

5.2.4.11 remove_matrix()

```
void remove_matrix (
    matrix_t * A )
```

Frees the memory allocated for the matrix.

This method frees the memory allocated for storing the matrix.

Parameters

<i>A</i>	Pointer to the matrix data structure to be deleted.
----------	---

5.2.4.12 setNewScale()

```
void setNewScale (
    data * drawing_data )
```

Sets a new model scale based on extreme values in the X, Y, and Z axes.

This method sets new model scale values based on extreme values along the X, Y, and Z axes.

Parameters

<i>drawing_data</i>	Pointer to the model data structure.
---------------------	--------------------------------------

5.2.4.13 setScaling()

```
void setScaling (
    const double minValue,
    const double maxValue,
    data * data_ )
```

Sets the scale to display the model.

This method sets the scale to display the model according to minimum and maximum coordinate values.

Parameters

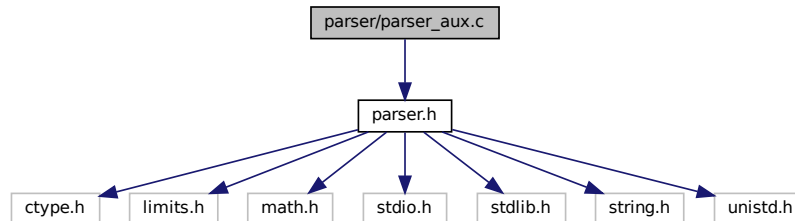
<i>minValue</i>	Minimum coordinate value.
<i>maxValue</i>	The maximum value of the coordinate.
<i>data_</i>	Pointer to the model data structure.

5.3 parser/parser_aux.c File Reference

Auxiliary file for file processing.

```
#include "parser.h"
```

Include dependency graph for parser_aux.c:



Functions

- int `create_matrix` (int rows, int columns, `matrix_t` *result)
Creates a matrix of the given dimensions.
- void `error_free` (`matrix_t` *mx, int index)
Frees the memory allocated for the matrix.
- void `remove_matrix` (`matrix_t` *A)
Frees the memory allocated for the matrix.
- void `check_mx_allocation` (double **mx, int *error)
*Checks if memory for matrix double *array was actually allocated.*
- void `check_row_allocation` (`matrix_t` *mx, int index, int *error)
Checks if memory for matrix double array was actually allocated.
- int `check_symbol` (const char ch, const char compCh)
Checks whether the character ch is equal to the character compCh.
- int `is_vertex` (char *buffer)
if current string contains vertex information
- int `is_facet` (char *buffer)
if current string contains facet information
- void `setScaling` (const double minValue, const double maxValue, `data` *data_)
Sets the scale to display the model.
- void `free_data` (`data` *drawing_data)
Frees memory occupied by the model data structure.
- void `setNewScale` (`data` *drawing_data)
Sets a new model scale based on extreme values in the X, Y, and Z axes.

5.3.1 Detailed Description

Auxiliary file for file processing.

Author

mitchelk, nenamaxi and dannamer

Version

0.1

Date

2024-03-22

Copyright

Copyright (c) 2024

5.3.2 Function Documentation

5.3.2.1 `check_mx_allocation()`

```
void check_mx_allocation (
    double ** mx,
    int * error )
```

Checks if memory for matrix double *array was actually allocated.

Parameters

<i>mx</i>	a pointer to a array of double * variables
<i>error</i>	a pointer to an error variable

5.3.2.2 `check_row_allocation()`

```
void check_row_allocation (
    matrix_t * mx,
    int index,
    int * error )
```

Checks if memory for matrix double array was actually allocated.

Parameters

<i>mx</i>	a matrix_t pointer
<i>index</i>	a number of allocated array elements
<i>error</i>	a pointer to an error code

5.3.2.3 check_symbol()

```
int check_symbol (
    const char ch,
    const char compCh )
```

Checks whether the character *ch* is equal to the character *compCh*.

This function compares the character *ch* with the character *compCh* and returns 1 if they are equal, or 0 otherwise.

Parameters

<i>ch</i>	The character to check.
<i>compCh</i>	The character to compare <i>ch</i> with.

Returns

Returns 1 if the characters are equal, 0 otherwise.

5.3.2.4 create_matrix()

```
int create_matrix (
    int rows,
    int columns,
    matrix_t * result )
```

Creates a matrix of the given dimensions.

This method creates a matrix with the specified number of rows and columns and initializes all its elements to zero.

Parameters

<i>rows</i>	Number of rows in the matrix.
<i>columns</i>	The number of columns in the matrix.
<i>result</i>	Pointer to the data structure into which the result of creating the matrix will be written.

Returns

int Matrix creation error code (PARSER_OK if successful).

5.3.2.5 error_free()

```
void error_free (
    matrix_t * mx,
    int index )
```

Frees the memory allocated for the matrix.

This function frees memory allocated for a matrix of type matrix_t.

Parameters

<i>mx</i>	Pointer to a matrix_t structure containing the matrix.
<i>index</i>	Index indicating the number of rows of the matrix.

5.3.2.6 free_data()

```
void free_data (
    data * drawing_data )
```

Frees memory occupied by the model data structure.

This method frees the memory allocated for storing the vertices and faces of the model.

Parameters

<i>drawing_data</i>	Pointer to the model data structure.
---------------------	--------------------------------------

5.3.2.7 is_facet()

```
int is_facet (
    char * buffer )
```

if current string contains facet information

Parameters

<i>buffer</i>	
---------------	--

Returns

1 -yes, 2 - no

5.3.2.8 is_vertex()

```
int is_vertex (
    char * buffer )
```

if current string contains vertex information

Parameters

<i>buffer</i>	
---------------	--

Returns

1 -yes, 2 - no

5.3.2.9 remove_matrix()

```
void remove_matrix (
    matrix_t * A )
```

Frees the memory allocated for the matrix.

This method frees the memory allocated for storing the matrix.

Parameters

<i>A</i>	Pointer to the matrix data structure to be deleted.
----------	---

5.3.2.10 setNewScale()

```
void setNewScale (
    data * drawing_data )
```

Sets a new model scale based on extreme values in the X, Y, and Z axes.

This method sets new model scale values based on extreme values along the X, Y, and Z axes.

Parameters

<i>drawing_data</i>	Pointer to the model data structure.
---------------------	--------------------------------------

5.3.2.11 setScaling()

```
void setScaling (
    const double minValue,
    const double maxValue,
    data * data_ )
```

Sets the scale to display the model.

This method sets the scale to display the model according to minimum and maximum coordinate values.

Parameters

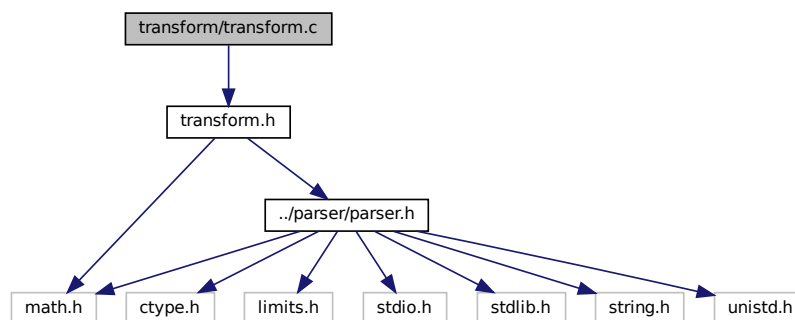
<i>minValue</i>	Minimum coordinate value.
<i>maxValue</i>	The maximum value of the coordinate.
<i>data_</i>	Pointer to the model data structure.

5.4 transform/transform.c File Reference

affine operations implementation file

```
#include "transform.h"
```

Include dependency graph for transform.c:



Functions

- void [rotation_by_ox](#) ([data](#) *drawing_data, const double valueCos, const double valueSin)

- void `rotation_by_oy` (`data *drawing_data`, `const double valueCos`, `const double valueSin`)
- void `rotation_by_oz` (`data *drawing_data`, `const double valueCos`, `const double valueSin`)
- int `getSign` (`const Rotation rotation_`)
Function to get sign.
- int `affineRotationOperation` (`data *data`, `const double angle`, `const Rotation rotation_`)
Affine rotation operation.
- int `affineMovingOperation` (`data *data`, `const double step`, `const Rotation rotation_`)
Affine moving operation.
- void `scaling` (`data *drawing_data`, `double factor`)
Helper function for performing the scaling operation.
- int `affineScalingOperation` (`data *data`, `const double coefficient`, `const Scaling scaling_`)
Affine scaling operation.
- void `setFigureToCenter` (`data *glData`)
Sets the figure to the center.

5.4.1 Detailed Description

affine operations implementation file

Author

mitchelk, nenamaxi and dannamer

Version

0.1

Date

2024-03-22

Copyright

Copyright (c) 2024

5.4.2 Function Documentation

5.4.2.1 `affineMovingOperation()`

```
int affineMovingOperation (  
    data * data,  
    const double step,  
    const Rotation rotation_ )
```

Affine moving operation.

Parameters

<i>data</i>	A <code>date</code> type structure for storing information about vertices.
<i>step</i>	Step of moving.
<i>rotation</i> \leftrightarrow	Argument of type "Rotation" to determine the direction.
—	

Returns

Returns the result of the operation: `ERROR` or `OK`.

See also

[getSign](#)

5.4.2.2 affineRotationOperation()

```
int affineRotationOperation (
    data * data,
    const double angle,
    const Rotation rotation_ )
```

Affine rotation operation.

Parameters

<i>data</i>	A <code>date</code> type structure for storing information about vertices.
<i>angle</i>	Angle of rotation.
<i>rotation</i> \leftrightarrow	Argument of type "Rotation" to determine the direction.
—	

Returns

Returns the result of the operation: `ERROR` or `OK`.

See also

[findCenterFigure](#)
[movingToPosition](#)
[getSign](#)
[rotation_by_ox](#)
[rotation_by_oy](#)
[rotation_by_oz](#)
[movingToPosition](#)

5.4.2.3 affineScalingOperation()

```
int affineScalingOperation (
    data * data,
    const double coefficient,
    const Scaling scaling_ )
```

Affine scaling operation.

Parameters

<i>data</i>	A data type structure for storing information about vertices.
<i>coefficient</i>	Scaling factor.
<i>scaling</i> ↔ —	Argument of type <code>Scaling</code> to define scaling(INCREASE or DECREASE).

Returns

Returns the result of the operation: `ERROR` or `OK`.

See also

[scaling](#)

5.4.2.4 getSign()

```
int getSign (
    const Rotation rotation_ )
```

Function to get sign.

Parameters

<i>rotation</i> ↔ —	Argument of type <code>Rotation</code> to determine the sign depending on the direction.
------------------------	--

Returns

sign.

5.4.2.5 rotation_by_ox()

```
void rotation_by_ox (
    data * drawing_data,
    const double valueCos,
    const double valueSin )
```

Parameters

<i>drawing_data</i>	Helper function for performing affine rotation operation around the Z
<i>valueCos</i>	Cosine of rotation angle.
<i>valueSin</i>	Sine of rotation angle.

5.4.2.6 rotation_by_oy()

```
void rotation_by_oy (
    data * drawing_data,
    const double valueCos,
    const double valueSin )
```

Parameters

<i>drawing_data</i>	Helper function for performing affine rotation operation around the Z
<i>valueCos</i>	Cosine of rotation angle.
<i>valueSin</i>	Sine of rotation angle.

5.4.2.7 rotation_by_oz()

```
void rotation_by_oz (
    data * drawing_data,
    const double valueCos,
    const double valueSin )
```

Parameters

<i>drawing_data</i>	Helper function for performing affine rotation operation around the Z
<i>valueCos</i>	Cosine of rotation angle.
<i>valueSin</i>	Sine of rotation angle.

5.4.2.8 scaling()

```
void scaling (
    data * drawing_data,
    double factor )
```

Helper function for performing the scaling operation.

Parameters

<i>drawing_data</i>	A <code>date</code> type structure for storing information about vertices.
<i>factor</i>	Scaling factor.

5.4.2.9 setFigureToCenter()

```
void setFigureToCenter (
    data * glData )
```

Sets the figure to the center.

Parameters

<i>glData</i>	A <code>date</code> type structure for storing information about vertices.
---------------	--

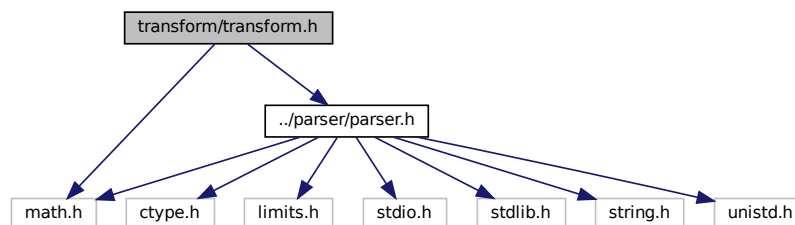
See also

[findCenterFigure](#)
[movingToPosition](#)
[setNewScale](#)

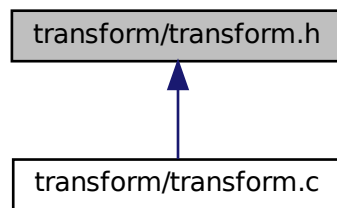
5.5 transform/transform.h File Reference

Header file of affine operations.

```
#include <math.h>
#include "../parser/parser.h"
Include dependency graph for transform.h:
```



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



Macros

- `#define OK 1`
Operation status macro.
- `#define ERROR 0`
Operation status macro.
- `#define ERROR_DIV_ZERO -1`
Operation status macro.
- `#define ST_MIN_SCALING 0.1`
Macro for minimal scaling.
- `#define ST_AROUND_VALUE_ONE 0.99`
Macro for around value one.

Enumerations

- enum `Scaling` { `INCREASE` = 0 , `DECREASE` = 1 }
- Enumeration structure for affine scaling operation.*
- enum `MOVING_TO_POSITION` { `CENTER` , `ORIGINAL` }
- Enumeration structure for operation moving.*

Functions

- void `scaling` (`data` *drawing_data, double factor)
Helper function for performing the scaling operation.
- void `rotation_by_ox` (`data` *drawing_data, const double valueCos, const double valueSin)
- void `rotation_by_oy` (`data` *drawing_data, const double valueCos, const double valueSin)
- void `rotation_by_oz` (`data` *drawing_data, const double valueCos, const double valueSin)
- int `getSign` (const `Rotation` rotation_)
Function to get sign.
- void `setFigureToCenter` (`data` *glData)
Sets the figure to the center.
- int `affineScalingOperation` (`data` *data, const double coefficient, const `Scaling` scaling)
Affine scaling operation.
- int `affineRotationOperation` (`data` *data, const double angle, const `Rotation` rotation_)
Affine rotation operation.
- int `affineMovingOperation` (`data` *data, const double step, const `Rotation` rotation_)
Affine moving operation.

5.5.1 Detailed Description

Header file of affine operations.

Author

mitchelk, nenamaxi and dannamer

Version

0.1

Date

2024-03-22

Copyright

Copyright (c) 2024

5.5.2 Enumeration Type Documentation

5.5.2.1 MOVING_TO_POSITION

enum [MOVING_TO_POSITION](#)

Enumeration structure for operation moving.

Enumerator

CENTER	Move to center
ORIGINAL	Move to start position

5.5.2.2 Scaling

enum [Scaling](#)

Enumeration structure for affine scaling operation.

Enumerator

INCREASE	Field indicating magnification
DECREASE	field indicating reduction

5.5.3 Function Documentation

5.5.3.1 affineMovingOperation()

```
int affineMovingOperation (
    data * data,
    const double step,
    const Rotation rotation_ )
```

Affine moving operation.

Parameters

<i>data</i>	A <code>date</code> type structure for storing information about vertices.
<i>step</i>	Step of moving.
<i>rotation</i> ↔	Argument of type "Rotation" to determine the direction.
—	

Returns

Returns the result of the operation: `ERROR` or `OK`.

See also

[getSign](#)

5.5.3.2 affineRotationOperation()

```
int affineRotationOperation (
    data * data,
    const double angle,
    const Rotation rotation_ )
```

Affine rotation operation.

Parameters

<i>data</i>	A <code>date</code> type structure for storing information about vertices.
<i>angle</i>	Angle of rotation.
<i>rotation</i> ↔	Argument of type "Rotation" to determine the direction.
—	

Returns

Returns the result of the operation: `ERROR` or `OK`.

See also

`findCenterFigure`
`movingToPosition`
[getSign](#)
[rotation_by_ox](#)
[rotation_by_oy](#)
[rotation_by_oz](#)
`movingToPosition`

5.5.3.3 `affineScalingOperation()`

```
int affineScalingOperation (
    data * data,
    const double coefficient,
    const Scaling scaling_ )
```

Affine scaling operation.

Parameters

<i>data</i>	A <code>data</code> type structure for storing information about vertices.
<i>coefficient</i>	Scaling factor.
<i>scaling</i> ↔	Argument of type <code>Scaling</code> to define scaling(<code>INCREASE</code> or <code>DECREASE</code>).
—	

Returns

Returns the result of the operation: `ERROR` or `OK`.

See also

[scaling](#)

5.5.3.4 `getSign()`

```
int getSign (
    const Rotation rotation_ )
```

Function to get sign.

Parameters

<i>rotation</i> ↔ —	Argument of type <code>Rotation</code> to determine the sign depending on the direction.
------------------------	--

Returns

sign.

5.5.3.5 `rotation_by_ox()`

```
void rotation_by_ox (
    data * drawing_data,
    const double valueCos,
    const double valueSin )
```

Parameters

<i>drawing_data</i>	Helper function for performing affine rotation operation around the Z
<i>valueCos</i>	Cosine of rotation angle.
<i>valueSin</i>	Sine of rotation angle.

5.5.3.6 `rotation_by_oy()`

```
void rotation_by_oy (
    data * drawing_data,
    const double valueCos,
    const double valueSin )
```

Parameters

<i>drawing_data</i>	Helper function for performing affine rotation operation around the Z
<i>valueCos</i>	Cosine of rotation angle.
<i>valueSin</i>	Sine of rotation angle.

5.5.3.7 `rotation_by_oz()`

```
void rotation_by_oz (
    data * drawing_data,
    const double valueCos,
    const double valueSin )
```

Parameters

<i>drawing_data</i>	Helper function for performing affine rotation operation around the Z
<i>valueCos</i>	Cosine of rotation angle.
<i>valueSin</i>	Sine of rotation angle.

5.5.3.8 scaling()

```
void scaling (
    data * drawing_data,
    double factor )
```

Helper function for performing the scaling operation.

Parameters

<i>drawing_data</i>	A <code>date</code> type structure for storing information about vertices.
<i>factor</i>	Scaling factor.

5.5.3.9 setFigureToCenter()

```
void setFigureToCenter (
    data * glData )
```

Sets the figure to the center.

Parameters

<i>glData</i>	A <code>date</code> type structure for storing information about vertices.
---------------	--

See also

`findCenterFigure`
`movingToPosition`
[setNewScale](#)

Index

affineMovingOperation
 transform.c, [27](#)
 transform.h, [34](#)
affineRotationOperation
 transform.c, [28](#)
 transform.h, [34](#)
affineScalingOperation
 transform.c, [28](#)
 transform.h, [35](#)

CENTER
 transform.h, [33](#)
check_mx_allocation
 parser.h, [16](#)
 parser_aux.c, [22](#)
check_row_allocation
 parser.h, [16](#)
 parser_aux.c, [22](#)
check_symbol
 parser.h, [16](#)
 parser_aux.c, [23](#)
columns
 matrix_struct, [10](#)
count_v
 facets, [9](#)
create_matrix
 parser.h, [17](#)
 parser_aux.c, [23](#)

Data, [7](#)
 facets_count, [8](#)
 gif_start, [8](#)
 polygons, [8](#)
 scale, [8](#)
 vertex_count, [8](#)
 vertexes, [8](#)

data
 parser.h, [15](#)

DECREASE
 transform.h, [33](#)

error_free
 parser.h, [17](#)
 parser_aux.c, [24](#)

facets, [9](#)
 count_v, [9](#)
 plnds, [9](#)

facets_count
 Data, [8](#)

free_data
 parser.h, [18](#)
 parser_aux.c, [24](#)

getSign
 transform.c, [29](#)
 transform.h, [35](#)

gif_start
 Data, [8](#)

INCREASE
 transform.h, [33](#)

is_facet
 parser.h, [18](#)
 parser_aux.c, [24](#)

is_vertex
 parser.h, [18](#)
 parser_aux.c, [25](#)

matrix
 matrix_struct, [10](#)
matrix_struct, [9](#)
 columns, [10](#)
 matrix, [10](#)
 rows, [10](#)

matrix_t
 parser.h, [15](#)
memory_handling
 parser.c, [12](#)
 parser.h, [19](#)

MOVING_TO_POSITION
 transform.h, [33](#)

ORIGINAL
 transform.h, [33](#)

parser
 parser.c, [12](#)
 parser.h, [19](#)

parser.c
 memory_handling, [12](#)
 parser, [12](#)

parser.h
 check_mx_allocation, [16](#)
 check_row_allocation, [16](#)
 check_symbol, [16](#)
 create_matrix, [17](#)
 data, [15](#)
 error_free, [17](#)
 free_data, [18](#)
 is_facet, [18](#)

- is_vertex, 18
- matrix_t, 15
- memory_handling, 19
- parser, 19
- polygon_t, 15
- remove_matrix, 19
- Rotation, 15
- setNewScale, 20
- setScaling, 20
- XM, 16
- XP, 16
- YM, 16
- YP, 16
- ZM, 16
- ZP, 16
- parser/parser.c, 11
- parser/parser.h, 13
- parser/parser_aux.c, 21
- parser_aux.c
 - check_mx_allocation, 22
 - check_row_allocation, 22
 - check_symbol, 23
 - create_matrix, 23
 - error_free, 24
 - free_data, 24
 - is_facet, 24
 - is_vertex, 25
 - remove_matrix, 25
 - setNewScale, 25
 - setScaling, 26
- plnds
 - facets, 9
- polygon_t
 - parser.h, 15
- polygons
 - Data, 8
- remove_matrix
 - parser.h, 19
 - parser_aux.c, 25
- Rotation
 - parser.h, 15
- rotation_by_ox
 - transform.c, 29
 - transform.h, 36
- rotation_by_oy
 - transform.c, 30
 - transform.h, 36
- rotation_by_oz
 - transform.c, 30
 - transform.h, 36
- rows
 - matrix_struct, 10
- scale
 - Data, 8
- Scaling
 - transform.h, 33
- scaling
 - transform.c, 30
 - transform.h, 37
- setFigureToCenter
 - transform.c, 31
 - transform.h, 37
- setNewScale
 - parser.h, 20
 - parser_aux.c, 25
- setScaling
 - parser.h, 20
 - parser_aux.c, 26
- transform.c
 - affineMovingOperation, 27
 - affineRotationOperation, 28
 - affineScalingOperation, 28
 - getSign, 29
 - rotation_by_ox, 29
 - rotation_by_oy, 30
 - rotation_by_oz, 30
 - scaling, 30
 - setFigureToCenter, 31
- transform.h
 - affineMovingOperation, 34
 - affineRotationOperation, 34
 - affineScalingOperation, 35
 - CENTER, 33
 - DECREASE, 33
 - getSign, 35
 - INCREASE, 33
 - MOVING_TO_POSITION, 33
 - ORIGINAL, 33
 - rotation_by_ox, 36
 - rotation_by_oy, 36
 - rotation_by_oz, 36
 - Scaling, 33
 - scaling, 37
 - setFigureToCenter, 37
- transform/transform.c, 26
- transform/transform.h, 31
- vertex_count
 - Data, 8
- vertexes
 - Data, 8
- XM
 - parser.h, 16
- XP
 - parser.h, 16
- YM
 - parser.h, 16
- YP
 - parser.h, 16
- ZM
 - parser.h, 16
- ZP
 - parser.h, 16