3D Viewer

Generated by Doxygen 1.9.1

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:

Chapter 3

File Index

3.1 File List

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

Chapter 4

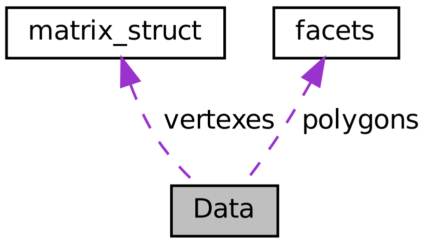
Class Documentation

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

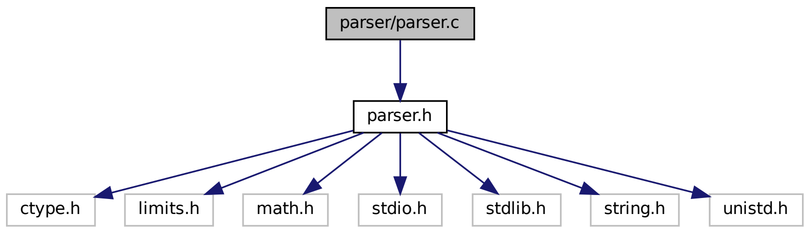
File Documentation

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

drawing_data	A `date` 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

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

Returns

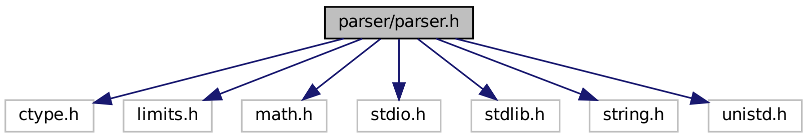
int Parsing error code (PARSER_OK if successful).

5.2 parser/parser.h File Reference

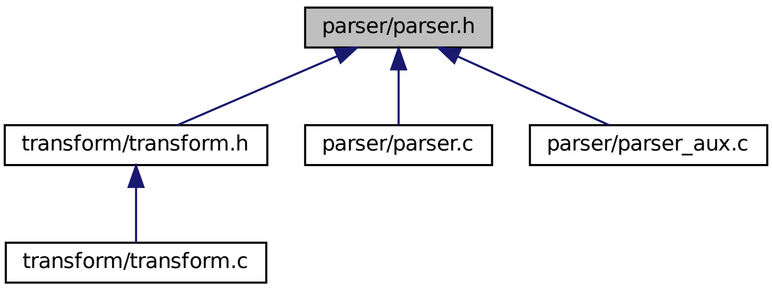
Header file for file processing.

```
#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

mx	a pointer to a array of double * varables
error	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

mx	a matrix_t pointer
index	a number of allocated array elements
error	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

ch	The character to check.
compCh	The character to compare ch 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

rows	Number of rows in the matrix.
columns	The number of columns in the matrix.
result	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

mx	Pointer to a matrix_t structure containing the matrix.
index	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

drawing_data	Pointer to the model data structure.

5.2.4.7 is_facet()

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

if current string contains facet information

Parameters

buffer	

Returns

1 -yes, 2 - no

5.2.4.8 is_vertex()

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

if current string contains vertex information

Parameters

buffer	

Returns

1 -yes, 2 - no

5.2.4.9 memory_handling()

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

Function for allocating memory for polygons.

Parameters

drawing_data	A `date` 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

drawing_data	Pointer to the data structure into which the parsing result will be written.
filename	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

A	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

drawing_data	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

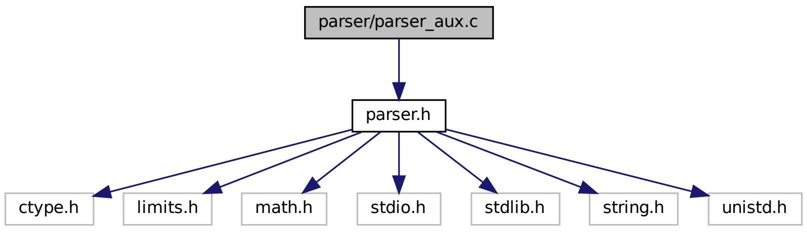
minValue	Minimum coordinate value.
maxValue	The maximum value of the coordinate.
*data_*	Pointer to the model data structure.

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

mx	a pointer to a array of double * varables
error	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

mx	a matrix_t pointer
index	a number of allocated array elements
error	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

ch	The character to check.
compCh	The character to compare ch 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

rows	Number of rows in the matrix.
columns	The number of columns in the matrix.
result	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

mx	Pointer to a matrix_t structure containing the matrix.
index	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

drawing_data	Pointer to the model data structure.

5.3.2.7 is_facet()

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

if current string contains facet information

Parameters

buffer	

Returns

1 -yes, 2 - no

5.3.2.8 is_vertex()

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

if current string contains vertex information

Parameters

buffer	

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

A	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

drawing_data	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.

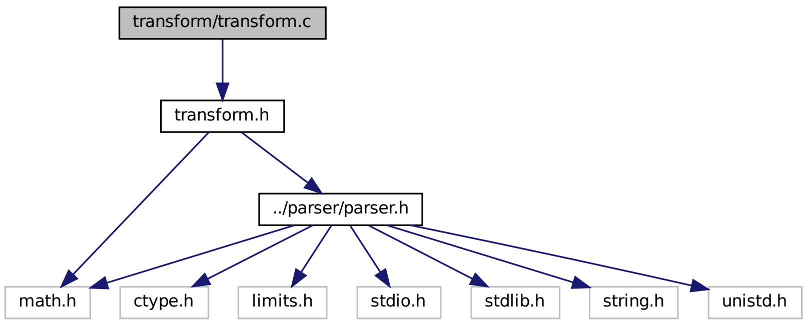
Parameters

minValue	Minimum coordinate value.
maxValue	The maximum value of the coordinate.
*data_*	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

data	A `date` type structure for storing information about vertices.
step	Step of moving.
*rotation_*	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

data	A `date` type structure for storing information about vertices.
angle	Angle of rotation.
*rotation_*	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

data	A `date` type structure for storing information about vertices.
coefficient	Scaling factor.
scaling↩ _	Argument of type `Scaling` 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

rotation↩ _	Argument of type `Rotation` 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

drawing_data	Helper function for performing affine rotation operation around the Z
valueCos	Cosine of rotation angle.
valueSin	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

drawing_data	Helper function for performing affine rotation operation around the Z
valueCos	Cosine of rotation angle.
valueSin	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

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

5.4.2.8 scaling()

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

Helper function for performing the scaling operation.

Parameters

drawing_data	A `date` type structure for storing information about vertices.
factor	Scaling factor.

5.4.2.9 setFigureToCenter()

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

Sets the figure to the center.

Parameters

glData	A `date` type structure for storing information about vertices.

See also

findCenterFigure

movingToPosition

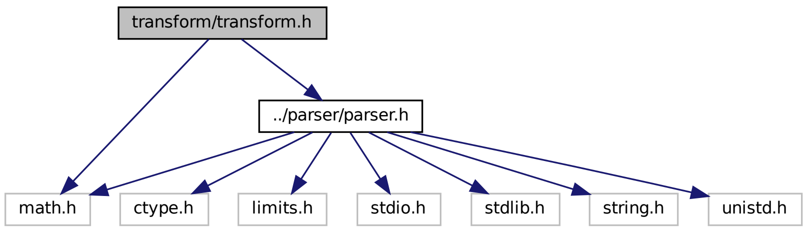
setNewScale

5.5 transform/transform.h File Reference

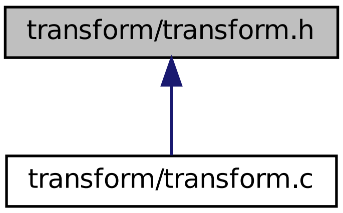
Header file of affine operations.

```
#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

data	A `date` type structure for storing information about vertices.
step	Step of moving.
rotation↩_	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

data	A `date` type structure for storing information about vertices.
angle	Angle of rotation.
rotation↩_	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

data	A `date` type structure for storing information about vertices.
coefficient	Scaling factor.
scaling↩ _	Argument of type `Scaling` to define scaling(`INCREASE` or `DECREASE`).

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

rotation←_	Argument of type `Rotation` 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

drawing_data	Helper function for performing affine rotation operation around the Z
valueCos	Cosine of rotation angle.
valueSin	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

drawing_data	Helper function for performing affine rotation operation around the Z
valueCos	Cosine of rotation angle.
valueSin	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

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

5.5.3.8 scaling()

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

Helper function for performing the scaling operation.

Parameters

drawing_data	A `date` type structure for storing information about vertices.
factor	Scaling factor.

5.5.3.9 setFigureToCenter()

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

Sets the figure to the center.

Parameters

glData	A `date` type structure for storing information about vertices.

See also

findCenterFigure

movingToPosition

setNewScale

Index