

CF\_CG-Lib

1.0

Generated by Doxygen 1.8.12



# Contents

<b>1</b>	<b>CF_CG-Lib</b>	<b>1</b>
<b>2</b>	<b>Namespace Index</b>	<b>3</b>
2.1	Namespace List . . . . .	3
<b>3</b>	<b>Hierarchical Index</b>	<b>5</b>
3.1	Class Hierarchy . . . . .	5
<b>4</b>	<b>Class Index</b>	<b>7</b>
4.1	Class List . . . . .	7
<b>5</b>	<b>File Index</b>	<b>9</b>
5.1	File List . . . . .	9
<b>6</b>	<b>Namespace Documentation</b>	<b>11</b>
6.1	cf Namespace Reference . . . . .	11
6.1.1	Typedef Documentation . . . . .	12
6.1.1.1	DirectionVector . . . . .	12
6.1.1.2	IFS . . . . .	12
6.1.1.3	LSystem . . . . .	12
6.1.1.4	ORB . . . . .	12
6.1.1.5	PointVector . . . . .	13
6.1.2	Function Documentation . . . . .	13
6.1.2.1	_removeWindowsSpecificCarriageReturn() . . . . .	13
6.1.2.2	degree2radian() . . . . .	13
6.1.2.3	radian2degree() . . . . .	13
6.1.2.4	readAntString() . . . . .	14
6.1.2.5	readDATFile() . . . . .	14
6.1.2.6	readPaletteFromFile() . . . . .	14

<b>7 Class Documentation</b>	<b>17</b>
7.1 cf::Color Struct Reference	17
7.1.1 Detailed Description	18
7.1.2 Constructor & Destructor Documentation	18
7.1.2.1 Color()	18
7.1.3 Member Function Documentation	18
7.1.3.1 invert()	18
7.1.3.2 operator!=()	18
7.1.3.3 operator*()	18
7.1.3.4 operator*=( )	18
7.1.3.5 operator+()	18
7.1.3.6 operator+=( )	19
7.1.3.7 operator-()	19
7.1.3.8 operator-=( )	19
7.1.3.9 operator/()	19
7.1.3.10 operator/=( )	19
7.1.3.11 operator==( )	19
7.1.4 Friends And Related Function Documentation	19
7.1.4.1 operator*	19
7.1.4.2 operator/	19
7.1.4.3 operator<<	20
7.1.5 Member Data Documentation	20
7.1.5.1 b	20
7.1.5.2 BLACK	20
7.1.5.3 BLUE	20
7.1.5.4 CYAN	20
7.1.5.5 g	20
7.1.5.6 GREEN	20
7.1.5.7 GREY	20
7.1.5.8 MAGENTA	20

7.1.5.9	ORANGE	21
7.1.5.10	PINK	21
7.1.5.11	r	21
7.1.5.12	RED	21
7.1.5.13	WHITE	21
7.1.5.14	YELLOW	21
7.2	cf::Console Struct Reference	21
7.2.1	Detailed Description	22
7.2.2	Member Function Documentation	22
7.2.2.1	clearConsole()	22
7.2.2.2	readFloat()	22
7.2.2.3	readInt()	22
7.2.2.4	readString()	22
7.2.2.5	waitKey()	23
7.3	cf::Direction Struct Reference	23
7.3.1	Detailed Description	23
7.3.2	Member Enumeration Documentation	23
7.3.2.1	AbsoluteDirection	23
7.3.2.2	RelativeDirection	24
7.3.3	Member Function Documentation	24
7.3.3.1	getNextiDirection()	24
7.3.3.2	toString() [1/2]	24
7.3.3.3	toString() [2/2]	24
7.4	cf::Interval Struct Reference	25
7.4.1	Detailed Description	25
7.4.2	Constructor & Destructor Documentation	25
7.4.2.1	Interval()	25
7.4.3	Member Function Documentation	25
7.4.3.1	translateIntervalPostion()	25
7.4.4	Friends And Related Function Documentation	26

7.4.4.1	<code>operator&lt;&lt;</code>	26
7.4.5	Member Data Documentation	26
7.4.5.1	<code>max</code>	26
7.4.5.2	<code>min</code>	26
7.5	<code>cf::IteratedFunctionSystem</code> Struct Reference	26
7.5.1	Detailed Description	26
7.5.2	Member Function Documentation	27
7.5.2.1	<code>getAllTransformation()</code>	27
7.5.2.2	<code>getName()</code>	27
7.5.2.3	<code>getNumTransformations()</code>	27
7.5.2.4	<code>getRangeX()</code>	27
7.5.2.5	<code>getRangeY()</code>	27
7.5.2.6	<code>getTransformation()</code>	27
7.5.2.7	<code>read()</code>	27
7.6	<code>cf::LindenmayerSystem</code> Struct Reference	28
7.6.1	Detailed Description	28
7.6.2	Member Function Documentation	28
7.6.2.1	<code>clearWindowEachTime()</code>	28
7.6.2.2	<code>getAdjustmentAngel()</code>	28
7.6.2.3	<code>getAllProductions()</code>	28
7.6.2.4	<code>getAxiom()</code>	28
7.6.2.5	<code>getName()</code>	29
7.6.2.6	<code>getNumProductions()</code>	29
7.6.2.7	<code>getProduction()</code>	29
7.6.2.8	<code>getRangeX()</code>	29
7.6.2.9	<code>getRangeY()</code>	29
7.6.2.10	<code>getScale()</code>	29
7.6.2.11	<code>getStartAngle()</code>	29
7.6.2.12	<code>read()</code>	29
7.7	<code>cf::Orbit</code> Struct Reference	30

7.7.1	Detailed Description	30
7.7.2	Member Function Documentation	30
7.7.2.1	getAllFactors()	30
7.7.2.2	getAllStartingPoints()	30
7.7.2.3	getName()	30
7.7.2.4	getNumFactors()	30
7.7.2.5	getNumStartingPoints()	30
7.7.2.6	getRangeX()	31
7.7.2.7	getRangeY()	31
7.7.2.8	read()	31
7.8	cf::Point Struct Reference	31
7.8.1	Detailed Description	32
7.8.2	Constructor & Destructor Documentation	32
7.8.2.1	Point()	32
7.8.3	Member Function Documentation	32
7.8.3.1	operator cv::Point()	32
7.8.3.2	operator"!=()	32
7.8.3.3	operator*()	32
7.8.3.4	operator*=(())	32
7.8.3.5	operator+()	33
7.8.3.6	operator+=(())	33
7.8.3.7	operator-()	33
7.8.3.8	operator-=(())	33
7.8.3.9	operator/()	33
7.8.3.10	operator/=(())	33
7.8.3.11	operator==(())	33
7.8.4	Friends And Related Function Documentation	33
7.8.4.1	operator*	33
7.8.4.2	operator/	34
7.8.5	Member Data Documentation	34

7.8.5.1	x	34
7.8.5.2	y	34
7.9	cf::Vec3< POINTVECTOR > Struct Template Reference	34
7.9.1	Detailed Description	35
7.9.2	Constructor & Destructor Documentation	36
7.9.2.1	Vec3() [1/3]	36
7.9.2.2	Vec3() [2/3]	36
7.9.2.3	Vec3() [3/3]	36
7.9.3	Member Function Documentation	36
7.9.3.1	getW()	36
7.9.3.2	getX()	36
7.9.3.3	getY()	37
7.9.3.4	isPointVector()	37
7.9.3.5	normalize()	37
7.9.3.6	operator cf::Point()	37
7.9.3.7	operator cf::Vec3< false >()	37
7.9.3.8	operator const glm::vec3 &()	37
7.9.3.9	operator glm::vec3()	38
7.9.3.10	operator%()	38
7.9.3.11	operator%=( )	38
7.9.3.12	operator*( ) [1/2]	38
7.9.3.13	operator*( ) [2/2]	39
7.9.3.14	operator*=( )	39
7.9.3.15	operator+( )	39
7.9.3.16	operator+=( )	39
7.9.3.17	operator-( )	39
7.9.3.18	operator-=( )	39
7.9.3.19	operator=( ) [1/2]	40
7.9.3.20	operator=( ) [2/2]	40
7.9.3.21	operator[]() [1/2]	40



7.9.3.22	<a href="#">operator[]()</a> [2/2]	40
7.9.3.23	<a href="#">setW()</a>	41
7.9.3.24	<a href="#">setX()</a>	41
7.9.3.25	<a href="#">setY()</a>	41
7.9.4	<a href="#">Friends And Related Function Documentation</a>	41
7.9.4.1	<a href="#">operator*</a>	41
7.9.4.2	<a href="#">operator&lt;&lt;()</a>	42
7.9.4.3	<a href="#">Vec3&lt;"!POINTVECTOR &gt;</a>	42
7.10	<a href="#">cf::Window2D Class Reference</a>	42
7.10.1	<a href="#">Detailed Description</a>	44
7.10.2	<a href="#">Member Enumeration Documentation</a>	44
7.10.2.1	<a href="#">LineType</a>	44
7.10.3	<a href="#">Constructor &amp; Destructor Documentation</a>	45
7.10.3.1	<a href="#">Window2D()</a> [1/2]	45
7.10.3.2	<a href="#">Window2D()</a> [2/2]	45
7.10.3.3	<a href="#">~Window2D()</a>	45
7.10.4	<a href="#">Member Function Documentation</a>	45
7.10.4.1	<a href="#">_convertFromNewInterval()</a>	45
7.10.4.2	<a href="#">_convertToNewInterval()</a>	45
7.10.4.3	<a href="#">_correctYValue()</a>	45
7.10.4.4	<a href="#">_window2foreground()</a>	45
7.10.4.5	<a href="#">clear()</a>	45
7.10.4.6	<a href="#">drawAxis()</a>	46
7.10.4.7	<a href="#">drawCircle()</a>	46
7.10.4.8	<a href="#">drawCriclePart()</a>	46
7.10.4.9	<a href="#">drawLine()</a>	47
7.10.4.10	<a href="#">drawRectangle()</a>	47
7.10.4.11	<a href="#">drawSpecializedLine()</a>	47
7.10.4.12	<a href="#">flippHorizontal()</a>	48
7.10.4.13	<a href="#">flippVertical()</a>	48

7.10.4.14 floodFill()	48
7.10.4.15 getColor()	48
7.10.4.16 getHeight()	49
7.10.4.17 getImage()	49
7.10.4.18 getIntervalX()	49
7.10.4.19 getIntervalY()	49
7.10.4.20 getInvertYAxis()	49
7.10.4.21 getWidth()	50
7.10.4.22 getWindowDisplayScale()	50
7.10.4.23 resetInterval()	50
7.10.4.24 resize()	50
7.10.4.25 saveImage()	50
7.10.4.26 setColor()	51
7.10.4.27 setInvertYAxis()	51
7.10.4.28 setNewInterval()	51
7.10.4.29 setWindowDisplayScale()	51
7.10.4.30 show()	51
7.10.4.31 waitKey()	52
7.10.4.32 waitMouseInput()	52
7.10.5 Member Data Documentation	52
7.10.5.1 m_FristShowCall	52
7.10.5.2 m_Image	52
7.10.5.3 m_IntervalChanged	52
7.10.5.4 m_IntervalX	52
7.10.5.5 m_IntervalY	53
7.10.5.6 m_InvertYAxis	53
7.10.5.7 m_MouseCallBackStorage	53
7.10.5.8 m_WindowName	53
7.10.5.9 m_WindowScale	53
7.11 cf::Window3D Struct Reference	53

7.11.1 Detailed Description . . . . .	55
7.11.2 Member Enumeration Documentation . . . . .	55
7.11.2.1 CameraType . . . . .	55
7.11.3 Constructor & Destructor Documentation . . . . .	55
7.11.3.1 Window3D() . . . . .	55
7.11.3.2 ~Window3D() . . . . .	56
7.11.4 Member Function Documentation . . . . .	56
7.11.4.1 _AdjustCamera() . . . . .	56
7.11.4.2 clear() . . . . .	56
7.11.4.3 disableLighting() . . . . .	56
7.11.4.4 draw() . . . . .	56
7.11.4.5 drawAxis() . . . . .	56
7.11.4.6 drawCylinder() [1/4] . . . . .	57
7.11.4.7 drawCylinder() [2/4] . . . . .	57
7.11.4.8 drawCylinder() [3/4] . . . . .	57
7.11.4.9 drawCylinder() [4/4] . . . . .	57
7.11.4.10 enableLighting() . . . . .	58
7.11.4.11 forceDisplay() . . . . .	58
7.11.4.12 getWindowHeight() . . . . .	58
7.11.4.13 getWindowWidth() . . . . .	58
7.11.4.14 handleKeyboardInput() . . . . .	58
7.11.4.15 printWindowUsage() . . . . .	58
7.11.4.16 setCamera() . . . . .	58
7.11.4.17 setMaxFPS() . . . . .	59
7.11.4.18 startDrawing() . . . . .	59
7.11.5 Friends And Related Function Documentation . . . . .	59
7.11.5.1 _DrawingFunction . . . . .	59
7.11.5.2 _KeyboardCallbackFunction . . . . .	59
7.11.6 Member Data Documentation . . . . .	60
7.11.6.1 m_AngleAdjustment . . . . .	60

7.11.6.2	<a href="#">m_CameraAdjustment</a>	60
7.11.6.3	<a href="#">m_CameraType</a>	60
7.11.6.4	<a href="#">m_DistAdjustment</a>	60
7.11.6.5	<a href="#">m_FreeCamera_LookDirection</a>	60
7.11.6.6	<a href="#">m_FreeCamera_position</a>	60
7.11.6.7	<a href="#">m_FreeCamera_UpVector</a>	60
7.11.6.8	<a href="#">m_LookAt</a>	60
7.11.6.9	<a href="#">m_LookAtDistance</a>	60
7.11.6.10	<a href="#">m_RotationAngle_X</a>	61
7.11.6.11	<a href="#">m_RotationAngle_Y</a>	61
7.12	<a href="#">cf::WindowCoordinateSystem Struct Reference</a>	61
7.12.1	<a href="#">Detailed Description</a>	62
7.12.2	<a href="#">Member Enumeration Documentation</a>	62
7.12.2.1	<a href="#">LineType</a>	62
7.12.3	<a href="#">Constructor &amp; Destructor Documentation</a>	62
7.12.3.1	<a href="#">WindowCoordinateSystem()</a>	62
7.12.3.2	<a href="#">~WindowCoordinateSystem()</a>	63
7.12.4	<a href="#">Member Function Documentation</a>	63
7.12.4.1	<a href="#">convert_intervalLength_to_pixellLength()</a>	63
7.12.4.2	<a href="#">convert_pixellLength_to_intervalLength()</a>	63
7.12.4.3	<a href="#">drawCircle()</a>	64
7.12.4.4	<a href="#">drawCriclePart()</a>	64
7.12.4.5	<a href="#">drawLine()</a>	64
7.12.4.6	<a href="#">drawLinearEquation()</a> [1/4]	65
7.12.4.7	<a href="#">drawLinearEquation()</a> [2/4]	65
7.12.4.8	<a href="#">drawLinearEquation()</a> [3/4]	66
7.12.4.9	<a href="#">drawLinearEquation()</a> [4/4]	66
7.12.4.10	<a href="#">drawPoint()</a>	66
7.12.4.11	<a href="#">setInterval()</a>	67
7.13	<a href="#">cf::WindowRasterized Struct Reference</a>	67

7.13.1 Detailed Description . . . . .	68
7.13.2 Member Enumeration Documentation . . . . .	68
7.13.2.1 LineType . . . . .	68
7.13.3 Constructor & Destructor Documentation . . . . .	68
7.13.3.1 WindowRasterized() [1/2] . . . . .	68
7.13.3.2 WindowRasterized() [2/2] . . . . .	68
7.13.3.3 ~WindowRasterized() . . . . .	69
7.14 cf::WindowVectorized Struct Reference . . . . .	69
7.14.1 Detailed Description . . . . .	70
7.14.2 Member Enumeration Documentation . . . . .	70
7.14.2.1 LineType . . . . .	70
7.14.3 Constructor & Destructor Documentation . . . . .	70
7.14.3.1 WindowVectorized() [1/2] . . . . .	70
7.14.3.2 WindowVectorized() [2/2] . . . . .	70
7.14.3.3 ~WindowVectorized() . . . . .	71
7.14.4 Member Function Documentation . . . . .	71
7.14.4.1 convert_intervalLength_to_pixelLength() . . . . .	71
7.14.4.2 convert_pixelLength_to_intervalLength() . . . . .	71
7.14.4.3 getColor_imageSpace() . . . . .	72
7.14.4.4 setColor_imageSpace() . . . . .	72
7.14.4.5 setInterval() . . . . .	72
7.14.4.6 transformPoint_fromImage_toInterval() . . . . .	73
7.14.4.7 transformPoint_fromInterval_toImage() . . . . .	73

<b>8 File Documentation</b>	<b>75</b>
8.1 include/computerGeometry.hpp File Reference	75
8.1.1 Function Documentation	76
8.1.1.1 operator<<()	76
8.2 include/IFS.h File Reference	76
8.3 include/LSystem.h File Reference	76
8.4 include/ORB.h File Reference	77
8.5 include/Utils.h File Reference	77
8.5.1 Function Documentation	78
8.5.1.1 operator<<() [1/5]	78
8.5.1.2 operator<<() [2/5]	78
8.5.1.3 operator<<() [3/5]	79
8.5.1.4 operator<<() [4/5]	79
8.5.1.5 operator<<() [5/5]	79
8.6 include/window2D.h File Reference	79
8.7 include/window3D.h File Reference	79
8.8 include/windowCoordinateSystem.hpp File Reference	80
8.9 include/windowRasterized.hpp File Reference	80
8.10 include/windowVercorized.hpp File Reference	80
8.11 README.md File Reference	81
<b>Index</b>	<b>83</b>

# Chapter 1

## CF\_CG-Lib

This library is intended to be used in 'Chaos und Fraktale' and 'Computer Geometry', lessons from 'Hochschule Darmstadt'. If you want to use it in a different way, you may do so under the terms of the MIT license.

The best way to find ALL functions is by going to 'namespaces [cf](#)' (Note: register 'classes' doesn't show 'namespace global' functions)

### The MIT License (MIT)

Copyright © 2016 Sascha Wombacher

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

### 3rd party licenses:

- GLM: MIT
- OpenCV: 3-clause BSD License
- InFLint: LGPL
- FreeGlut: X-Consortium

Note: OpenCV utilizes 3rd party libraries like libpng, these licenses are NOT covered in this segment





## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">cf</a>	.....	<a href="#">11</a>
--------------------	-------	--------------------



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

cf::Color . . . . .	17
cf::Console . . . . .	21
cf::Direction . . . . .	23
cf::Interval . . . . .	25
cf::IteratedFunctionSystem . . . . .	26
cf::LindenmayerSystem . . . . .	28
cf::Orbit . . . . .	30
cf::Point . . . . .	31
cf::Vec3< POINTVECTOR > . . . . .	34
cf::Window2D . . . . .	42
cf::WindowCoordinateSystem . . . . .	61
cf::WindowRasterized . . . . .	67
cf::WindowVectorized . . . . .	69
cf::Window3D . . . . .	53



## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">cf::Color</a>	The <a href="#">Color</a> struct offers a class for rgb access . . . . .	17
<a href="#">cf::Console</a>	The <a href="#">Console</a> struct offers utility functions for 'console' . . . . .	21
<a href="#">cf::Direction</a>	The <a href="#">Direction</a> struct for getting absolute directions from a current direction and a relative direction	23
<a href="#">cf::Interval</a>	The <a href="#">Interval</a> struct provides functionality to translate values from one interval into another . . .	25
<a href="#">cf::IteratedFunctionSystem</a>	The <a href="#">IteratedFunctionSystem</a> class lazy people (like myself) may use the IFS typedef . . . . .	26
<a href="#">cf::LindenmayerSystem</a>	The <a href="#">LindenmayerSystem</a> class lazy people (like myself) may use the IFS typedef . . . . .	28
<a href="#">cf::Orbit</a>	The <a href="#">Orbit</a> class lazy people (like myself) may use the ORB typedef . . . . .	30
<a href="#">cf::Point</a>	The <a href="#">Point</a> struct is a simple class for position access on 2D images (imilar to cv::Point, but uses floats instead of integer) . . . . .	31
<a href="#">cf::Vec3&lt; POINTVECTOR &gt;</a>	The <a href="#">Vec3</a> struct General class for vector operations . . . . .	34
<a href="#">cf::Window2D</a>	The <a href="#">Window2D</a> struct offers advanced features used by WindowRasterized/WindowVectorized	42
<a href="#">cf::Window3D</a>	The <a href="#">Window3D</a> struct is the default class for accessing 3D content, creating more than 1 instance results in undefined behavior . . . . .	53
<a href="#">cf::WindowCoordinateSystem</a>	The <a href="#">WindowCoordinateSystem</a> struct Default class for images and raster operations . . . . .	61
<a href="#">cf::WindowRasterized</a>	The <a href="#">WindowRasterized</a> struct Default struct for vectorized operations within a custom interval	67
<a href="#">cf::WindowVectorized</a>	The <a href="#">WindowVectorized</a> struct Default class for images and raster operations . . . . .	69



## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

include/computerGeometry.hpp . . . . .	75
include/IFS.h . . . . .	76
include/LSystem.h . . . . .	76
include/ORB.h . . . . .	77
include/utils.h . . . . .	77
include/window2D.h . . . . .	79
include/window3D.h . . . . .	79
include/windowCoordinateSystem.hpp . . . . .	80
include/windowRasterized.hpp . . . . .	80
include/windowVercorized.hpp . . . . .	80





## Chapter 6

# Namespace Documentation

### 6.1 cf Namespace Reference

#### Classes

- struct [Color](#)  
*The [Color](#) struct offers a class for rgb access.*
- struct [Console](#)  
*The [Console](#) struct offers utility functions for 'console'.*
- struct [Direction](#)  
*The [Direction](#) struct for getting absolute directions from a current direction and a relative direction.*
- struct [Interval](#)  
*The [Interval](#) struct provides functionality to translate values from one interval into another.*
- struct [IteratedFunctionSystem](#)  
*The [IteratedFunctionSystem](#) class lazy people (like myself) may use the IFS typedef.*
- struct [LindenmayerSystem](#)  
*The [LindenmayerSystem](#) class lazy people (like myself) may use the IFS typedef.*
- struct [Orbit](#)  
*The [Orbit](#) class lazy people (like myself) may use the ORB typedef.*
- struct [Point](#)  
*The [Point](#) struct is a simple class for position access on 2D images (similar to cv::Point, but uses floats instead of integer)*
- struct [Vec3](#)  
*The [Vec3](#) struct General class for vector operations.*
- class [Window2D](#)  
*The [Window2D](#) struct offers advanced features used by WindowRasterized/WindowVectorized.*
- struct [Window3D](#)  
*The [Window3D](#) struct is the default class for accessing 3D content, creating more than 1 instance results in undefined behavior.*
- struct [WindowCoordinateSystem](#)  
*The [WindowCoordinateSystem](#) struct Default class for images and raster operations.*
- struct [WindowRasterized](#)  
*The [WindowRasterized](#) struct Default struct for vectorized operations within a custom interval.*
- struct [WindowVectorized](#)  
*The [WindowVectorized](#) struct Default class for images and raster operations.*

## Typedefs

- typedef [Vec3](#)< true > [PointVector](#)  
*PointVector Specialiazion of general [Vec3](#).*
- typedef [Vec3](#)< false > [DirectionVector](#)  
*DirectionVector Specialiazion of general [Vec3](#), where component 'w' may not be written to.*
- typedef [IteratedFunctionSystem](#) IFS
- typedef [LindenmayerSystem](#) LSystem
- typedef [Orbit](#) ORB

## Functions

- void [\\_removeWindowsSpecificCarriageReturn](#) (std::string &str)  
*\_removeWindowsSpecificCarriageReturn Removes 'carriage return' characters in strings ('carriage return' may be read from unix system by providing windows files)*
- std::vector< [Color](#) > [readPaletteFromFile](#) (const std::string &filePath)  
*readPaletteFromFile*
- std::string [readAntString](#) (const std::string &filePath)  
*readAntString*
- template<typename \_VectorType = glm::vec3>  
std::vector< \_VectorType > [readDATFile](#) (const std::string &filePath)  
*readDATFile Reads a \*.dat file*
- float [radian2degree](#) (float radianValue)  
*radian2degree Converts a radian value to a degree value*
- float [degree2radian](#) (float degreeValue)  
*degree2radian Converts a degree value to a radian value*

### 6.1.1 Typedef Documentation

#### 6.1.1.1 DirectionVector

```
typedef Vec3<false> cf::DirectionVector
```

[DirectionVector](#) Specialiazion of general [Vec3](#), where component 'w' may not be written to.

#### 6.1.1.2 IFS

```
typedef IteratedFunctionSystem cf::IFS
```

#### 6.1.1.3 LSystem

```
typedef LindenmayerSystem cf::LSystem
```

#### 6.1.1.4 ORB

```
typedef Orbit cf::ORB
```

### 6.1.1.5 PointVector

```
typedef Vec3<true > cf::PointVector
```

PointVector Specialiazion of general [Vec3](#).

## 6.1.2 Function Documentation

### 6.1.2.1 \_removeWindowsSpecificCarriageReturn()

```
void cf::_removeWindowsSpecificCarriageReturn (
    std::string & str )
```

`_removeWindowsSpecificCarriageReturn` Removes 'carriage return' characters in strings ('carriage return' may be read from unix system by providing windows files)

#### Parameters

<i>str</i>	string containing 'carriage return', which will be removed
------------	--

### 6.1.2.2 degree2radian()

```
float cf::degree2radian (
    float degreeValue )
```

`degree2radian` Converts a degree value to a radian value

#### Parameters

<i>degreeValue</i>	Degree value to be converted
--------------------	------------------------------

#### Returns

Converted radian value

### 6.1.2.3 radian2degree()

```
float cf::radian2degree (
    float radianValue )
```

`radian2degree` Converts a radian value to a degree value

#### Parameters

<i>radianValue</i>	Radian value to be converted
--------------------	------------------------------

**Returns**

Converted degree value

**6.1.2.4 readAntString()**

```
std::string cf::readAntString (
    const std::string & filePath )
```

readAntString

**Parameters**

<i>filePath</i>	Read *.ant file from path
-----------------	---------------------------

**Returns****6.1.2.5 readDATFile()**

```
template<typename _VectorType = glm::vec3>
std::vector<_VectorType> cf::readDATFile (
    const std::string & filePath )
```

readDATFile Reads a \*.dat file

**Parameters**

<i>filePath</i>	Read *.dat file from path
-----------------	---------------------------

**Returns****6.1.2.6 readPaletteFromFile()**

```
std::vector<Color> cf::readPaletteFromFile (
    const std::string & filePath )
```

readPaletteFromFile

**Parameters**

<i>filePath</i>	Read *.pal file from path
-----------------	---------------------------

Returns



## Chapter 7

# Class Documentation

### 7.1 cf::Color Struct Reference

The `Color` struct offers a class for rgb access.

```
#include <utils.h>
```

#### Public Member Functions

- `Color` (uint8\_t red=0, uint8\_t green=0, uint8\_t blue=0)
- `Color operator*` (float value)
- `Color operator/` (float value)
- `Color & operator*=` (float value)
- `Color & operator/=` (float value)
- `Color operator+` (const `Color` &c)
- `Color operator-` (const `Color` &c)
- `Color & operator+=` (const `Color` &c)
- `Color & operator-=` (const `Color` &c)
- `bool operator==` (const `Color` &c)
- `bool operator!=` (const `Color` &c)
- `Color invert` () const

#### Public Attributes

- uint8\_t `b`
- uint8\_t `g`
- uint8\_t `r`

#### Static Public Attributes

- static const `Color MAGENTA`
- static const `Color YELLOW`
- static const `Color ORANGE`
- static const `Color WHITE`
- static const `Color BLACK`
- static const `Color GREEN`
- static const `Color GREY`
- static const `Color BLUE`
- static const `Color CYAN`
- static const `Color PINK`
- static const `Color RED`

## Friends

- `cf::Color operator*` (float value, const `cf::Color` &c)
- `cf::Color operator/` (float value, const `cf::Color` &c)
- `std::ostream & operator<<` (`std::ostream` &os, const `Color` &c)

### 7.1.1 Detailed Description

The `Color` struct offers a class for rgb access.

### 7.1.2 Constructor & Destructor Documentation

#### 7.1.2.1 `Color()`

```
cf::Color::Color (
    uint8_t red = 0,
    uint8_t green = 0,
    uint8_t blue = 0 ) [inline]
```

### 7.1.3 Member Function Documentation

#### 7.1.3.1 `invert()`

```
Color cf::Color::invert ( ) const
```

#### 7.1.3.2 `operator!=()`

```
bool cf::Color::operator!= (
    const Color & c )
```

#### 7.1.3.3 `operator*()`

```
Color cf::Color::operator* (
    float value )
```

#### 7.1.3.4 `operator*=( )`

```
Color& cf::Color::operator*= (
    float value )
```

#### 7.1.3.5 `operator+()`

```
Color cf::Color::operator+ (
    const Color & c )
```



#### 7.1.3.6 operator+=()

```
Color& cf::Color::operator+= (
    const Color & c )
```

#### 7.1.3.7 operator-()

```
Color cf::Color::operator- (
    const Color & c )
```

#### 7.1.3.8 operator-=()

```
Color& cf::Color::operator-= (
    const Color & c )
```

#### 7.1.3.9 operator/()

```
Color cf::Color::operator/ (
    float value )
```

#### 7.1.3.10 operator/=()

```
Color& cf::Color::operator/= (
    float value )
```

#### 7.1.3.11 operator==(())

```
bool cf::Color::operator== (
    const Color & c )
```

### 7.1.4 Friends And Related Function Documentation

#### 7.1.4.1 operator\*

```
cf::Color operator* (
    float value,
    const cf::Color & c ) [friend]
```

#### 7.1.4.2 operator/

```
cf::Color operator/ (
    float value,
    const cf::Color & c ) [friend]
```

#### 7.1.4.3 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const Color & c ) [friend]
```

### 7.1.5 Member Data Documentation

#### 7.1.5.1 b

```
uint8_t cf::Color::b
```

#### 7.1.5.2 BLACK

```
const Color cf::Color::BLACK [static]
```

#### 7.1.5.3 BLUE

```
const Color cf::Color::BLUE [static]
```

#### 7.1.5.4 CYAN

```
const Color cf::Color::CYAN [static]
```

#### 7.1.5.5 g

```
uint8_t cf::Color::g
```

#### 7.1.5.6 GREEN

```
const Color cf::Color::GREEN [static]
```

#### 7.1.5.7 GREY

```
const Color cf::Color::GREY [static]
```

#### 7.1.5.8 MAGENTA

```
const Color cf::Color::MAGENTA [static]
```

### 7.1.5.9 ORANGE

```
const Color cf::Color::ORANGE [static]
```

### 7.1.5.10 PINK

```
const Color cf::Color::PINK [static]
```

### 7.1.5.11 r

```
uint8_t cf::Color::r
```

### 7.1.5.12 RED

```
const Color cf::Color::RED [static]
```

### 7.1.5.13 WHITE

```
const Color cf::Color::WHITE [static]
```

### 7.1.5.14 YELLOW

```
const Color cf::Color::YELLOW [static]
```

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

- include/[utils.h](#)

## 7.2 cf::Console Struct Reference

The [Console](#) struct offers utility functions for 'console'.

```
#include <utils.h>
```

### Static Public Member Functions

- static std::string [readString](#) ()  
*readString* Read a line into a std::string (includes spaces)
- static float [readFloat](#) ()  
*readFloat* Reads a floatingpoint value
- static int [readInt](#) ()  
*readInt* Reads a integer value
- static void [waitKey](#) ()  
*waitKey* Wait until key input (on windows also sets the console window active)
- static void [clearConsole](#) ()  
*clearConsole* Clears the console

### 7.2.1 Detailed Description

The [Console](#) struct offers utility functions for 'console'.

### 7.2.2 Member Function Documentation

#### 7.2.2.1 clearConsole()

```
static void cf::Console::clearConsole ( ) [static]
```

clearConsole Clears the console

#### 7.2.2.2 readFloat()

```
static float cf::Console::readFloat ( ) [static]
```

readFloat Reads a floatingpoint value

##### Returns

Read value

#### 7.2.2.3 readInt()

```
static int cf::Console::readInt ( ) [static]
```

readInt Reads a integer value

##### Returns

Read value

#### 7.2.2.4 readString()

```
static std::string cf::Console::readString ( ) [static]
```

readString Read a line into a std::string (includes spaces)

##### Returns

Read line

## 7.2.2.5 waitKey()

```
static void cf::Console::waitKey ( ) [static]
```

waitKey Wait until key input (on windows also sets the console window active)

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

- include/Utils.h

## 7.3 cf::Direction Struct Reference

The [Direction](#) struct for getting absolute directions from a current direction and a relative direction.

```
#include <utils.h>
```

### Public Types

- enum [AbsoluteDirection](#) { [AbsoluteDirection::NORTH](#), [AbsoluteDirection::EAST](#), [AbsoluteDirection::SOUTH](#), [AbsoluteDirection::WEST](#), [AbsoluteDirection::NUM\\_ABS\\_DIRS](#) }
- enum [RelativeDirection](#) { [RelativeDirection::LEFT](#), [RelativeDirection::FORWARD](#), [RelativeDirection::RIGHT](#), [RelativeDirection::NUM\\_REL\\_DIRS](#) }

### Static Public Member Functions

- static [AbsoluteDirection](#) getNextDirection ([AbsoluteDirection](#) currentDirection, [RelativeDirection](#) relativeMovement)  
*getNextDirection receive absolute direction by providing a relative direction*
- static std::string toString ([AbsoluteDirection](#) absDir)
- static std::string toString ([RelativeDirection](#) relDir)

#### 7.3.1 Detailed Description

The [Direction](#) struct for getting absolute directions from a current direction and a relative direction.

#### 7.3.2 Member Enumeration Documentation

##### 7.3.2.1 AbsoluteDirection

```
enum cf::Direction::AbsoluteDirection [strong]
```

##### Enumerator

NORTH	
EAST	
SOUTH	
WEST	
NUM_ABS_DIRS	

### 7.3.2.2 RelativeDirection

```
enum cf::Direction::RelativeDirection [strong]
```

#### Enumerator

LEFT	
FORWARD	
RIGHT	
NUM_REL_DIRS	

## 7.3.3 Member Function Documentation

### 7.3.3.1 getNextiDirection()

```
static AbsoluteDirection cf::Direction::getNextiDirection (
    AbsoluteDirection currentDirection,
    RelativeDirection relativeMovement ) [static]
```

getNextiDirection receive absolute direction by providing a relative directon

#### Parameters

<i>currentDirection</i>	current absolute direction
<i>relativeMovement</i>	relative movement

#### Returns

### 7.3.3.2 toString() [1/2]

```
static std::string cf::Direction::toString (
    AbsoluteDirection absDir ) [static]
```

### 7.3.3.3 toString() [2/2]

```
static std::string cf::Direction::toString (
    RelativeDirection relDir ) [static]
```

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

- [include/utils.h](#)

## 7.4 cf::Interval Struct Reference

The [Interval](#) struct provides functionality to translate values from one interval into another.

```
#include <utils.h>
```

### Public Member Functions

- [Interval](#) (float `_min`=0, float `_max`=0)

### Static Public Member Functions

- static float [translateIntervalPostion](#) (const [Interval](#) &originalInterval, const [Interval](#) &newInterval, float originalPosition)

### Public Attributes

- float [min](#)
- float [max](#)

### Friends

- std::ostream & [operator<<](#) (std::ostream &os, const [Interval](#) &interval)

#### 7.4.1 Detailed Description

The [Interval](#) struct provides functionality to translate values from one interval into another.

#### 7.4.2 Constructor & Destructor Documentation

##### 7.4.2.1 Interval()

```
cf::Interval::Interval (
    float _min = 0,
    float _max = 0 ) [inline]
```

#### 7.4.3 Member Function Documentation

##### 7.4.3.1 translateIntervalPostion()

```
static float cf::Interval::translateIntervalPostion (
    const Interval & originalInterval,
    const Interval & newInterval,
    float originalPosition ) [static]
```

## 7.4.4 Friends And Related Function Documentation

### 7.4.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const Interval & interval ) [friend]
```

## 7.4.5 Member Data Documentation

### 7.4.5.1 max

```
float cf::Interval::max
```

### 7.4.5.2 min

```
float cf::Interval::min
```

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

- [include/utils.h](#)

## 7.5 cf::IteratedFunctionSystem Struct Reference

The [IteratedFunctionSystem](#) class lazy people (like myself) may use the IFS typedef.

```
#include <IFS.h>
```

### Public Member Functions

- void [read](#) (const std::string &fiilePath)  
*read a \*.ifs file from path*
- std::size\_t [getNumTransformations](#) () const
- const glm::mat3x3 & [getTransformation](#) (std::size\_t pos) const
- const [Interval](#) & [getRangeX](#) () const
- const [Interval](#) & [getRangeY](#) () const
- const std::string & [getName](#) () const
- const std::vector< glm::mat3x3 > & [getAllTransformation](#) () const

### 7.5.1 Detailed Description

The [IteratedFunctionSystem](#) class lazy people (like myself) may use the IFS typedef.



## 7.5.2 Member Function Documentation

### 7.5.2.1 getAllTransformation()

```
const std::vector<glm::mat3x3>& cf::IteratedFunctionSystem::getAllTransformation ( ) const
```

### 7.5.2.2 getName()

```
const std::string& cf::IteratedFunctionSystem::getName ( ) const
```

### 7.5.2.3 getNumTransformations()

```
std::size_t cf::IteratedFunctionSystem::getNumTransformations ( ) const
```

### 7.5.2.4 getRangeX()

```
const Interval& cf::IteratedFunctionSystem::getRangeX ( ) const
```

### 7.5.2.5 getRangeY()

```
const Interval& cf::IteratedFunctionSystem::getRangeY ( ) const
```

### 7.5.2.6 getTransformation()

```
const glm::mat3x3& cf::IteratedFunctionSystem::getTransformation (
    std::size_t pos ) const
```

### 7.5.2.7 read()

```
void cf::IteratedFunctionSystem::read (
    const std::string & fiilePath )
```

read a \*.ifs file from path

#### Parameters

<i>fiilePath</i>	Path to a *.ifs file
------------------	----------------------

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

- [include/IFS.h](#)

## 7.6 cf::LindenmayerSystem Struct Reference

The [LindenmayerSystem](#) class lazy people (like myself) may use the IFS tyepdef.

```
#include <LSystem.h>
```

### Public Member Functions

- void [read](#) (const std::string &filePath)  
*read a \*.lin file from path*
- const std::string & [getName](#) () const
- char [getAxiom](#) () const
- const std::string \* [getProduction](#) (char symbol) const
- std::size\_t [getNumProductions](#) () const
- bool [clearWindowEachTime](#) () const
- const [Interval](#) & [getRangeX](#) () const
- const [Interval](#) & [getRangeY](#) () const
- float [getScale](#) () const
- float [getStartAngle](#) () const
- float [getAdjustmentAngel](#) () const
- const std::vector< std::pair< const char, const std::string > > & [getAllProductions](#) () const

### 7.6.1 Detailed Description

The [LindenmayerSystem](#) class lazy people (like myself) may use the IFS tyepdef.

### 7.6.2 Member Function Documentation

#### 7.6.2.1 clearWindowEachTime()

```
bool cf::LindenmayerSystem::clearWindowEachTime ( ) const
```

#### 7.6.2.2 getAdjustmentAngel()

```
float cf::LindenmayerSystem::getAdjustmentAngel ( ) const
```

#### 7.6.2.3 getAllProductions()

```
const std::vector<std::pair<const char, const std::string> >& cf::LindenmayerSystem::getAll↵  
Productions ( ) const
```

#### 7.6.2.4 getAxiom()

```
char cf::LindenmayerSystem::getAxiom ( ) const
```

### 7.6.2.5 getName()

```
const std::string& cf::LindenmayerSystem::getName ( ) const
```

### 7.6.2.6 getNumProductions()

```
std::size_t cf::LindenmayerSystem::getNumProductions ( ) const
```

### 7.6.2.7 getProduction()

```
const std::string* cf::LindenmayerSystem::getProduction (
    char symbol ) const
```

### 7.6.2.8 getRangeX()

```
const Interval& cf::LindenmayerSystem::getRangeX ( ) const
```

### 7.6.2.9 getRangeY()

```
const Interval& cf::LindenmayerSystem::getRangeY ( ) const
```

### 7.6.2.10 getScale()

```
float cf::LindenmayerSystem::getScale ( ) const
```

### 7.6.2.11 getStartAngle()

```
float cf::LindenmayerSystem::getStartAngle ( ) const
```

### 7.6.2.12 read()

```
void cf::LindenmayerSystem::read (
    const std::string & filePath )
```

read a \*.lin file from path

#### Parameters

<i>filePath</i>	Path to a *.lin file
-----------------	----------------------

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

- [include/LSystem.h](#)

## 7.7 cf::Orbit Struct Reference

The [Orbit](#) class lazy people (like myself) may use the ORB typedef.

```
#include <ORB.h>
```

### Public Member Functions

- void [read](#) (const std::string &filePath)  
*read a \*.orb file from path*
- const [Interval](#) & [getRangeX](#) () const
- const [Interval](#) & [getRangeY](#) () const
- const std::string & [getName](#) () const
- const std::vector< glm::vec3 > & [getAllStartingPoints](#) () const
- const std::vector< float > & [getAllFactors](#) () const
- std::size\_t [getNumFactors](#) () const
- std::size\_t [getNumStartingPoints](#) () const

#### 7.7.1 Detailed Description

The [Orbit](#) class lazy people (like myself) may use the ORB typedef.

#### 7.7.2 Member Function Documentation

##### 7.7.2.1 getAllFactors()

```
const std::vector<float>& cf::Orbit::getAllFactors ( ) const
```

##### 7.7.2.2 getAllStartingPoints()

```
const std::vector<glm::vec3>& cf::Orbit::getAllStartingPoints ( ) const
```

##### 7.7.2.3 getName()

```
const std::string& cf::Orbit::getName ( ) const
```

##### 7.7.2.4 getNumFactors()

```
std::size_t cf::Orbit::getNumFactors ( ) const
```

##### 7.7.2.5 getNumStartingPoints()

```
std::size_t cf::Orbit::getNumStartingPoints ( ) const
```

## 7.7.2.6 getRangeX()

```
const Interval& cf::Orbit::getRangeX ( ) const
```

## 7.7.2.7 getRangeY()

```
const Interval& cf::Orbit::getRangeY ( ) const
```

## 7.7.2.8 read()

```
void cf::Orbit::read (
    const std::string & filePath )
```

read a \*.orb file from path

## Parameters

<i>filePath</i>	Path to a *.orb file
-----------------	----------------------

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

- include/ORB.h

## 7.8 cf::Point Struct Reference

The [Point](#) struct is a simple class for position access on 2D images (imilar to cv::Point, but uses floats instead of integer)

```
#include <window2D.h>
```

## Public Member Functions

- [Point](#) (float val\_x=0.f, float val\_y=0.f)
- bool [operator==](#) (const [Point](#) &p) const
- bool [operator!=](#) (const [Point](#) &p) const
- [Point operator+](#) (const [Point](#) &p) const
- [Point & operator+=](#) (const [Point](#) &p)
- [Point operator-](#) (const [Point](#) &p) const
- [Point & operator-=](#) (const [Point](#) &p)
- [Point operator\\*](#) (float factor) const
- [Point & operator\\*=](#) (float factor)
- [Point operator/](#) (float rhs) const
- [Point & operator/=](#) (float rhs)
- [operator cv::Point](#) () const

## Public Attributes

- float `x`
- float `y`

## Friends

- `Point operator*` (float lhs, const `Point` &p)
- `Point operator/` (float lhs, const `Point` &p)

## 7.8.1 Detailed Description

The `Point` struct is a simple class for position access on 2D images (imilar to `cv::Point`, but uses floats instead of integer)

## 7.8.2 Constructor & Destructor Documentation

### 7.8.2.1 `Point()`

```
cf::Point::Point (
    float val_x = 0.f,
    float val_y = 0.f ) [inline]
```

## 7.8.3 Member Function Documentation

### 7.8.3.1 `operator cv::Point()`

```
cf::Point::operator cv::Point ( ) const
```

### 7.8.3.2 `operator!=()`

```
bool cf::Point::operator!= (
    const Point & p ) const
```

### 7.8.3.3 `operator*()`

```
Point cf::Point::operator* (
    float factor ) const
```

### 7.8.3.4 `operator*=( )`

```
Point& cf::Point::operator*= (
    float factor )
```

#### 7.8.3.5 operator+()

```
Point cf::Point::operator+ (
    const Point & p ) const
```

#### 7.8.3.6 operator+=()

```
Point& cf::Point::operator+= (
    const Point & p )
```

#### 7.8.3.7 operator-()

```
Point cf::Point::operator- (
    const Point & p ) const
```

#### 7.8.3.8 operator-=()

```
Point& cf::Point::operator-= (
    const Point & p )
```

#### 7.8.3.9 operator/()

```
Point cf::Point::operator/ (
    float rhs ) const
```

#### 7.8.3.10 operator/=()

```
Point& cf::Point::operator/= (
    float rhs )
```

#### 7.8.3.11 operator==(())

```
bool cf::Point::operator== (
    const Point & p ) const
```

### 7.8.4 Friends And Related Function Documentation

#### 7.8.4.1 operator\*

```
Point operator* (
    float lhs,
    const Point & p ) [friend]
```

#### 7.8.4.2 operator/

```
Point operator/ (
    float lhs,
    const Point & p ) [friend]
```

### 7.8.5 Member Data Documentation

#### 7.8.5.1 x

```
float cf::Point::x
```

#### 7.8.5.2 y

```
float cf::Point::y
```

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

- include/window2D.h

## 7.9 cf::Vec3< POINTVECTOR > Struct Template Reference

The [Vec3](#) struct General class for vector operations.

```
#include <computerGeometry.hpp>
```

### Public Member Functions

- [Vec3](#) (float x=0.f, float y=0.f)
- [Vec3](#) (float x, float y, float w)
- [Vec3](#) (const [cf::Point](#) &p)
- template<bool RHS>  
[Vec3](#)< RHS|POINTVECTOR > [operator+](#) (const [Vec3](#)< RHS > &rhs) const
- template<bool RHS>  
[Vec3](#)< POINTVECTOR > & [operator+=](#) (const [Vec3](#)< RHS > &rhs)
- template<bool RHS>  
[Vec3](#)< RHS|POINTVECTOR > [operator-](#) (const [Vec3](#)< RHS > &rhs) const
- template<bool RHS>  
[Vec3](#)< POINTVECTOR > & [operator-=](#) (const [Vec3](#)< RHS > &rhs)
- [cf::Vec3](#)< POINTVECTOR > [operator\\*](#) (float rhs) const  
*operator\* Multiplies each component of the vector with a factor*
- [cf::Vec3](#)< POINTVECTOR > & [operator\\*=](#) (float rhs)
- template<bool RHS>  
[Vec3](#)< RHS|POINTVECTOR > [operator%](#) (const [Vec3](#)< RHS > &rhs) const  
*operator% Performs the cross product between two vectors*
- template<bool RHS>  
[Vec3](#)< POINTVECTOR > & [operator%=>](#) (const [Vec3](#)< RHS > &rhs)
- void [normalize](#) ()



- normalize* Normalizes the PointVector (division by the 'w' component), compile error on DirectionVecotrs
- bool `isPointVector` () const  
*isPointVector* Checks wether a Vector is a PointVector or DirectionVector
- template<bool RHS>  
float `operator*` (const `Vec3`< RHS > &rhs) const  
*operator\** Performs the dot product between two vectors
- float `getX` () const  
*getX* Read access to component 'x'
- float `getY` () const  
*getY* Read access to component 'y'
- float `getW` () const  
*getW* Read access to component 'w'
- void `setX` (float value)  
*setX* Write to component 'x'
- void `setY` (float value)  
*setY* Write to component 'y'
- void `setW` (float value)  
*setW* Write to component 'w', compile error on DirectionVectors
- float `operator[]` (int idx) const  
*operator[]* Access to each component of the Vector, Note: read access is granted to all components (including index 2)
- float & `operator[]` (int idx)  
*operator[]* Access to each component of the Vector, Note: no write access for index 2 on DirectionVectors
- `operator glm::vec3` () const
- `operator const glm::vec3 &` () const
- `operator cf::Point` () const  
*operator cf::Point* Conversion operator to `cf::Point`, compile error on DirectionVectors
- `cf::PointVector & operator=` (const `cf::Point` &p)
- `cf::Vec3`< POINTVECTOR > & `operator=` (const glm::vec3 &rhs)
- `operator cf::Vec3`< false > () const  
*operator cf::DirectionVector* Conversion operator from PointVector to DirectionVector, exception if 'w' is not 0

## Friends

- struct `Vec3`<!POINTVECTOR >
- `cf::Vec3`< POINTVECTOR > `operator*` (float lhs, const `cf::Vec3`< POINTVECTOR > &vec)
- template<bool b>  
std::ostream & `operator<<` (std::ostream &, const `Vec3`< b > &)

### 7.9.1 Detailed Description

```
template<bool POINTVECTOR>
struct cf::Vec3< POINTVECTOR >
```

The `Vec3` struct General class for vector operations.

it porvieds:

- conversions from/to `cf::Point` and `glm::vec3`
- Cross product ('operator') and dot product ('operator\*') with other vectors
- Support for DirectionVectors and PointVectors (see typedef 'PointVector' and 'DirectionVector')

## 7.9.2 Constructor & Destructor Documentation

### 7.9.2.1 Vec3() [1/3]

```
template<bool POINTVECTOR>
cf::Vec3< POINTVECTOR >::Vec3 (
    float x = 0.f,
    float y = 0.f ) [inline]
```

### 7.9.2.2 Vec3() [2/3]

```
template<bool POINTVECTOR>
cf::Vec3< POINTVECTOR >::Vec3 (
    float x,
    float y,
    float w ) [inline]
```

### 7.9.2.3 Vec3() [3/3]

```
template<bool POINTVECTOR>
cf::Vec3< POINTVECTOR >::Vec3 (
    const cf::Point & p ) [inline]
```

## 7.9.3 Member Function Documentation

### 7.9.3.1 getW()

```
template<bool POINTVECTOR>
float cf::Vec3< POINTVECTOR >::getW ( ) const [inline]
```

getW Read access to component 'w'

**Returns**

### 7.9.3.2 getX()

```
template<bool POINTVECTOR>
float cf::Vec3< POINTVECTOR >::getX ( ) const [inline]
```

getX Read access to component 'x'

**Returns**

## 7.9.3.3 getY()

```
template<bool POINTVECTOR>
float cf::Vec3< POINTVECTOR >::getY ( ) const [inline]
```

getY Read access to component 'y'

Returns

## 7.9.3.4 isPointVector()

```
template<bool POINTVECTOR>
bool cf::Vec3< POINTVECTOR >::isPointVector ( ) const [inline]
```

isPointVector Checks wether a Vector is a PointVector or DirectionVector

Returns

## 7.9.3.5 normalize()

```
template<bool POINTVECTOR>
void cf::Vec3< POINTVECTOR >::normalize ( ) [inline]
```

normalize Normalizes the PointVector (division by the 'w' component), compile error on DirectionVecotrs

## 7.9.3.6 operator cf::Point()

```
template<bool POINTVECTOR>
cf::Vec3< POINTVECTOR >::operator cf::Point ( ) const [inline]
```

operator cf::Point Conversion operator to cf::Point, compile error on DirectionVectors

## 7.9.3.7 operator cf::Vec3&lt; false &gt;()

```
template<bool POINTVECTOR>
cf::Vec3< POINTVECTOR >::operator cf::Vec3< false > ( ) const [inline]
```

operator cf::DirectionVector Conversion operator from PointVector to DirectionVector, exception if 'w' is not 0

## 7.9.3.8 operator const glm::vec3 &amp;()

```
template<bool POINTVECTOR>
cf::Vec3< POINTVECTOR >::operator const glm::vec3 & ( ) const [inline]
```

### 7.9.3.9 operator glm::vec3()

```
template<bool POINTVECTOR>
cf::Vec3< POINTVECTOR >::operator glm::vec3 ( ) const [inline]
```

### 7.9.3.10 operator%()

```
template<bool POINTVECTOR>
template<bool RHS>
Vec3<RHS | POINTVECTOR> cf::Vec3< POINTVECTOR >::operator% (
    const Vec3< RHS > & rhs ) const [inline]
```

operator% Performs the cross product between two vectors

#### Parameters

<i>rhs</i>	Second operand for cross product
------------	----------------------------------

#### Returns

### 7.9.3.11 operator%=( )

```
template<bool POINTVECTOR>
template<bool RHS>
Vec3<POINTVECTOR>& cf::Vec3< POINTVECTOR >::operator%= (
    const Vec3< RHS > & rhs ) [inline]
```

### 7.9.3.12 operator\*( ) [1/2]

```
template<bool POINTVECTOR>
cf::Vec3<POINTVECTOR> cf::Vec3< POINTVECTOR >::operator* (
    float rhs ) const [inline]
```

operator\* Multiplies each component of the vector with a factor

#### Parameters

<i>rhs</i>	Factor for the multiplication
------------	-------------------------------

#### Returns

Multiplied vector

**7.9.3.13 operator\*()** [2/2]

```
template<bool POINTVECTOR>
template<bool RHS>
float cf::Vec3< POINTVECTOR >::operator* (
    const Vec3< RHS > & rhs ) const [inline]
```

**operator\*** Performs the dot product between two vectors

**Parameters**

<i>rhs</i>	Second operand for dot product
------------	--------------------------------

**Returns****7.9.3.14 operator\*=( )**

```
template<bool POINTVECTOR>
cf::Vec3<POINTVECTOR>& cf::Vec3< POINTVECTOR >::operator*= (
    float rhs ) [inline]
```

**7.9.3.15 operator+( )**

```
template<bool POINTVECTOR>
template<bool RHS>
Vec3<RHS | POINTVECTOR> cf::Vec3< POINTVECTOR >::operator+ (
    const Vec3< RHS > & rhs ) const [inline]
```

**7.9.3.16 operator+=( )**

```
template<bool POINTVECTOR>
template<bool RHS>
Vec3<POINTVECTOR>& cf::Vec3< POINTVECTOR >::operator+= (
    const Vec3< RHS > & rhs ) [inline]
```

**7.9.3.17 operator-( )**

```
template<bool POINTVECTOR>
template<bool RHS>
Vec3<RHS | POINTVECTOR> cf::Vec3< POINTVECTOR >::operator- (
    const Vec3< RHS > & rhs ) const [inline]
```

**7.9.3.18 operator-=( )**

```
template<bool POINTVECTOR>
template<bool RHS>
Vec3<POINTVECTOR>& cf::Vec3< POINTVECTOR >::operator-= (
    const Vec3< RHS > & rhs ) [inline]
```

**7.9.3.19 operator=()** [1/2]

```
template<bool POINTVECTOR>
cf::PointVector& cf::Vec3< POINTVECTOR >::operator= (
    const cf::Point & p ) [inline]
```

**7.9.3.20 operator=()** [2/2]

```
template<bool POINTVECTOR>
cf::Vec3<POINTVECTOR>& cf::Vec3< POINTVECTOR >::operator= (
    const glm::vec3 & rhs ) [inline]
```

**7.9.3.21 operator[]()** [1/2]

```
template<bool POINTVECTOR>
float cf::Vec3< POINTVECTOR >::operator[] (
    int idx ) const [inline]
```

**operator[]** Access to each component of the Vector, Note: read access is granted to all components (including index 2)

**Parameters**

<i>idx</i>	Access index
------------	--------------

**Returns****7.9.3.22 operator[]()** [2/2]

```
template<bool POINTVECTOR>
float& cf::Vec3< POINTVECTOR >::operator[] (
    int idx ) [inline]
```

**operator[]** Access to each component of the Vector, Note: no write access for index 2 on DirectionVectors

**Parameters**

<i>idx</i>	Access index, idx = 0 -> x, idx = 1 -> y, idx = 2 -> w
------------	--

**Returns**

## 7.9.3.23 setW()

```
template<bool POINTVECTOR>
void cf::Vec3< POINTVECTOR >::setW (
    float value ) [inline]
```

setW Write to component 'w', compile error on DirectionVectors

## Parameters

<i>value</i>	
--------------	--

## 7.9.3.24 setX()

```
template<bool POINTVECTOR>
void cf::Vec3< POINTVECTOR >::setX (
    float value ) [inline]
```

setX Write to component 'x'

## Parameters

<i>value</i>	
--------------	--

## 7.9.3.25 setY()

```
template<bool POINTVECTOR>
void cf::Vec3< POINTVECTOR >::setY (
    float value ) [inline]
```

setY Write to component 'y'

## Parameters

<i>value</i>	
--------------	--

## 7.9.4 Friends And Related Function Documentation

## 7.9.4.1 operator\*

```
template<bool POINTVECTOR>
cf::Vec3<POINTVECTOR> operator* (
    float lhs,
    const cf::Vec3< POINTVECTOR > & vec ) [friend]
```

#### 7.9.4.2 operator<<)

```
template<bool POINTVECTOR>
template<bool b>
std::ostream& operator<< (> (
    std::ostream & ,
    const Vec3<b> & ) [friend]
```

#### 7.9.4.3 Vec3<!POINTVECTOR >

```
template<bool POINTVECTOR>
friend struct Vec3<!POINTVECTOR > [friend]
```

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

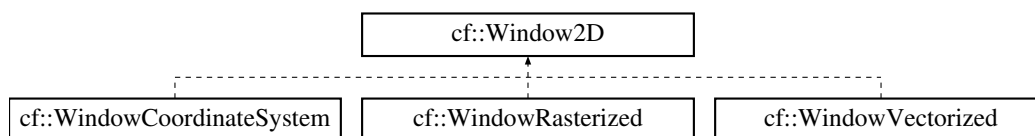
- [include/computerGeometry.hpp](#)

## 7.10 cf::Window2D Class Reference

The [Window2D](#) struct offers advanced features used by WindowRasterized/WindowVectorized.

```
#include <window2D.h>
```

Inheritance diagram for cf::Window2D:



### Public Types

- enum [LineType](#) {  
[LineType::DEFAULT](#) = 0, [LineType::DOT\\_0](#) = Window2D::DOT\_VALUE | 1, [LineType::DOT\\_1](#), [LineType::DOT\\_2](#),  
[LineType::DASH\\_0](#) = Window2D::DASH\_VALUE | 1, [LineType::DASH\\_1](#), [LineType::DASH\\_2](#), [LineType::DOT\\_DASH\\_0](#) = Window2D::DOT\_VALUE | Window2D::DASH\_VALUE | 1,  
[LineType::DOT\\_DASH\\_1](#), [LineType::DOT\\_DASH\\_2](#) }

*The LineType enum Special line type used by one function of 'drawLine'.*



## Public Member Functions

- [Window2D](#) (int width=800, int height=600, const char \*windowName="Chaos and Fractals", const [cf::Color](#) &startColor={0, 0, 0})
- [Window2D](#) (const char \*filePath)
- virtual [~Window2D](#) ()
- void [show](#) () const  
*show Show image, on first call it may require additional time to display content correctly (in those cases use wait←Key(1000) )*
- void [clear](#) (const [cf::Color](#) &color=[cf::Color::WHITE](#))
- unsigned char [waitKey](#) (int delay=0) const  
*waitKey Block access until key input on window*
- void [waitMouseInput](#) (float &x, float &y)  
*waitMouseInput Blocks until mouse input has been given*
- void [setWindowDisplayScale](#) (float scale)  
*setWindowDisplayScale Scales the image before displaying*
- float [getWindowDisplayScale](#) () const
- void [setInvertYAxis](#) (bool invert)  
*setInvertYAxis Invert y values on all 'cf::Point' functions*
- bool [getInvertYAxis](#) () const
- void [setColor](#) (float x, float y, const [Color](#) &color)
- [Color](#) [getColor](#) (float x, float y) const
- void [drawCircle](#) ([cf::Point](#) center, int radius, int lineWidth, const [cf::Color](#) &color)  
*drawCircle Draws a circle around the center*
- void [drawRectangle](#) ([cf::Point](#) point1, [cf::Point](#) point2, int lineWidth, const [cf::Color](#) &color)  
*drawRectangle Draws a rectangle from two diagonal points*
- void [drawLine](#) ([cf::Point](#) point1, [cf::Point](#) point2, int lineWidth, const [cf::Color](#) &color)  
*drawLine Draws a line from point1 to point2*
- void [drawSpecializedLine](#) ([cf::Point](#) point1, [cf::Point](#) point2, [LineType](#) lineType, const [cf::Color](#) &color)  
*drawSpecializedLine Draws specialized line of width 1 (dotted and/or dashed lines)*
- void [setNewInterval](#) (const [cf::Interval](#) &intervalX, const [cf::Interval](#) &intervalY)  
*setNewInterval Set new interval*
- void [resetInterval](#) ()  
*resetInterval Set default interval (interval x: [0, image width - 1], interval y: [0, image height - 1])*
- void [saveImage](#) (const char \*filePath) const  
*saveImage Saves current image to harddrive*
- void [resize](#) (int pixelWidth, int pixelHeight)  
*resize Resize underlying image*
- void [flippHorizontal](#) ()  
*flippHorizontal Flipp image horizontally*
- void [flippVertical](#) ()  
*flippHorizontal Flipp image vertically*
- const [cf::Interval](#) & [getIntervalX](#) () const  
*getIntervalX Const access to interval in x direction*
- const [cf::Interval](#) & [getIntervalY](#) () const  
*getIntervalY Const access to interval in y direction*
- int [getWidth](#) () const  
*getWidth Access to underlying image width*
- int [getHeight](#) () const  
*getHeight Access to underlying image height*
- [cv::Mat](#) & [getImage](#) ()  
*getImage Direct access to the underlying image*

- void [drawAxis](#) (const [cf::Color](#) &color=[cf::Color::BLACK](#), float stepSize\_x=1.f, float stepSize\_y=1.f, float interceptLength=3.f)  
*drawAxis This function draws x and y axis based on [Interval](#)*
- void [drawCriclePart](#) ([cf::Point](#) center, int radius, float startAngle, float endAngle, int lineWidth, const [cf::Color](#) &color=[cf::Color::BLACK](#))  
*drawCriclePart Draws a part of a circle*
- void [floodFill](#) ([cf::Point](#) startingPoint, const [cf::Color](#) &color)  
*floodFill Fills an area*

## Protected Member Functions

- void [\\_correctYValue](#) (float &y) const
- void [\\_convertFromNewInterval](#) (float &x, float &y) const
- void [\\_convertToNewInterval](#) (float &x, float &y) const
- void [\\_window2foreground](#) () const

## Protected Attributes

- cv::Mat [m\\_Image](#)
- bool [m\\_InvertYAxis](#)
- const char \* [m\\_WindowName](#)
- float [m\\_WindowScale](#)
- [cf::Interval](#) [m\\_IntervalX](#)
- [cf::Interval](#) [m\\_IntervalY](#)
- float [m\\_MouseCallBackStorage](#) [2]
- bool [m\\_IntervalChanged](#) = false
- bool [m\\_FristShowCall](#) = true

### 7.10.1 Detailed Description

The [Window2D](#) struct offers advanced features used by WindowRasterized/WindowVertorized.

### 7.10.2 Member Enumeration Documentation

#### 7.10.2.1 LineType

```
enum cf::Window2D::LineType [strong]
```

The LineType enum Special line type used by one function of 'drawLine'.

#### Enumerator

DEFAULT	
DOT_0	
DOT_1	
DOT_2	
DASH_0	
DASH_1	
DASH_2	
DOT_DASH↔ _0	
DOT_DASH↔ _1	
DOT_DASH↔ _2	

## 7.10.3 Constructor & Destructor Documentation

### 7.10.3.1 Window2D() [1/2]

```
cf::Window2D::Window2D (
    int width = 800,
    int height = 600,
    const char * windowName = "Chaos and Fractals",
    const cf::Color & startColor = {0, 0, 0} )
```

### 7.10.3.2 Window2D() [2/2]

```
cf::Window2D::Window2D (
    const char * filePath )
```

### 7.10.3.3 ~Window2D()

```
virtual cf::Window2D::~~Window2D ( ) [virtual]
```

## 7.10.4 Member Function Documentation

### 7.10.4.1 \_convertFromNewInterval()

```
void cf::Window2D::_convertFromNewInterval (
    float & x,
    float & y ) const [protected]
```

### 7.10.4.2 \_convertToNewInterval()

```
void cf::Window2D::_convertToNewInterval (
    float & x,
    float & y ) const [protected]
```

### 7.10.4.3 \_correctYValue()

```
void cf::Window2D::_correctYValue (
    float & y ) const [protected]
```

### 7.10.4.4 \_window2foreground()

```
void cf::Window2D::_window2foreground ( ) const [protected]
```

### 7.10.4.5 clear()

```
void cf::Window2D::clear (
    const cf::Color & color = cf::Color::WHITE )
```

## 7.10.4.6 drawAxis()

```
void cf::Window2D::drawAxis (
    const cf::Color & color = cf::Color::BLACK,
    float stepSize_x = 1.f,
    float stepSize_y = 1.f,
    float interceptLength = 3.f )
```

drawAxis This function draws x and y axis based on [Interval](#)

## Parameters

<i>color</i>	Axis color, default is white
<i>stepSize</i> ↔ <i>_x</i>	Dynamially set step size (x-axis), negative numbers indicate 10 steps for interval x
<i>stepSize</i> ↔ <i>_y</i>	Dynamially set step size (y-axis), negative numbers indicate 10 steps for interval y

## 7.10.4.7 drawCircle()

```
void cf::Window2D::drawCircle (
    cf::Point center,
    int radius,
    int lineWidth,
    const cf::Color & color )
```

drawCircle Draws a circle around the center

## Parameters

<i>point</i>	<a href="#">Point</a> within interval_x and interval_y
<i>radius</i>	Circle radius in pixel (not effected by intervals)
<i>lineWidth</i>	Pixelwidth of line (not effected by intervals)
<i>color</i>	Circle color

## 7.10.4.8 drawCriclePart()

```
void cf::Window2D::drawCriclePart (
    cf::Point center,
    int radius,
    float startAngle,
    float endAngle,
    int lineWidth,
    const cf::Color & color = cf::Color::BLACK )
```

drawCriclePart Draws a part of a circle

## Parameters

<i>center</i>	Center point of the circle
---------------	----------------------------

## Parameters

<i>radius</i>	Radius of the circle
<i>startAngle</i>	Start position (in degrees)
<i>endAngle</i>	End position (in degrees)
<i>color</i>	Color of the drawn line

## 7.10.4.9 drawLine()

```
void cf::Window2D::drawLine (
    cf::Point point1,
    cf::Point point2,
    int lineWidth,
    const cf::Color & color )
```

drawLine Draws a line from point1 to point2

## Parameters

<i>point1</i>	Point within interval_x and interval_y
<i>point2</i>	Point within interval_x and interval_y
<i>lineWidth</i>	Line width in pixel size
<i>color</i>	Line color

## 7.10.4.10 drawRectangle()

```
void cf::Window2D::drawRectangle (
    cf::Point point1,
    cf::Point point2,
    int lineWidth,
    const cf::Color & color )
```

drawRectangle Draws a rectangle from two diagonal points

## Parameters

<i>point1</i>	Point within interval_x and interval_y, has to be the diagonal point to point2
<i>point2</i>	Point within interval_x and interval_y, has to be the diagonal point to point1
<i>lineWidth</i>	LineWidth pixelwidth of line (not effected by intervals)
<i>color</i>	Rectangle color

## 7.10.4.11 drawSpecializedLine()

```
void cf::Window2D::drawSpecializedLine (
    cf::Point point1,
```

```

    cf::Point point2,
    LineType lineType,
    const cf::Color & color )

```

drawSpecializedLine Draws specialized line of width 1 (dotted and/or dashed lines)

#### Parameters

<i>point1</i>	Point within interval_x and interval_y
<i>point2</i>	Point within interval_x and interval_y
<i>lineType</i>	Type of line to be drawn
<i>color</i>	Line color

#### 7.10.4.12 flippHorizontal()

```
void cf::Window2D::flippHorizontal ( )
```

flippHorizontal Flipp image horizontally

#### 7.10.4.13 flippVertical()

```
void cf::Window2D::flippVertical ( )
```

flippHorizontal Flipp image vertically

#### 7.10.4.14 floodFill()

```

void cf::Window2D::floodFill (
    cf::Point startingPoint,
    const cf::Color & color )

```

floodFill Fills an area

#### Parameters

<i>startingPoint</i>	First point to be colored
<i>color</i>	Fill color

#### 7.10.4.15 getColor()

```

Color cf::Window2D::getColor (
    float x,
    float y ) const

```

#### 7.10.4.16 getHeight()

```
int cf::Window2D::getHeight ( ) const
```

getHeight Access to underlying image height

##### Returns

Height

#### 7.10.4.17 getImage()

```
cv::Mat& cf::Window2D::getImage ( )
```

getImage Direct access to the underlying image

##### Returns

Image handle

#### 7.10.4.18 getIntervalX()

```
const cf::Interval& cf::Window2D::getIntervalX ( ) const
```

getIntervalX Const access to interval in x direction

##### Returns

#### 7.10.4.19 getIntervalY()

```
const cf::Interval& cf::Window2D::getIntervalY ( ) const
```

getIntervalY Const access to interval in y direction

##### Returns

#### 7.10.4.20 getInvertYAxis()

```
bool cf::Window2D::getInvertYAxis ( ) const
```

#### 7.10.4.21 getWidth()

```
int cf::Window2D::getWidth ( ) const
```

getWidth Access to underlying image width

##### Returns

Width

#### 7.10.4.22 getWindowDisplayScale()

```
float cf::Window2D::getWindowDisplayScale ( ) const
```

#### 7.10.4.23 resetInterval()

```
void cf::Window2D::resetInterval ( )
```

resetInterval Set default interval (interval x: [0, image width - 1], interval y: [0, image height - 1])

#### 7.10.4.24 resize()

```
void cf::Window2D::resize (
    int pixelWidth,
    int pixelHeight )
```

resize Resize underlying image

##### Parameters

<i>pixelWidth</i>	New width
<i>pixelHeight</i>	New height

#### 7.10.4.25 saveImage()

```
void cf::Window2D::saveImage (
    const char * filePath ) const
```

saveImage Saves current image to harddrive

##### Parameters

<i>filePath</i>	File path and name, format will be determined based on file ending (*.png, *.jpeg, ...)
-----------------	---



## 7.10.4.26 setColor()

```
void cf::Window2D::setColor (
    float x,
    float y,
    const Color & color )
```

## 7.10.4.27 setInvertYAxis()

```
void cf::Window2D::setInvertYAxis (
    bool invert )
```

setInvertYAxis Invert y values on all 'cf::Point' functions

## Parameters

<i>invert</i>	
---------------	--

## 7.10.4.28 setNewInterval()

```
void cf::Window2D::setNewInterval (
    const cf::Interval & intervalX,
    const cf::Interval & intervalY )
```

setNewInterval Set new interval

## Parameters

<i>intervalX</i>	Interval in x direction
<i>intervalY</i>	Interval in y direction

## 7.10.4.29 setWindowDisplayScale()

```
void cf::Window2D::setWindowDisplayScale (
    float scale )
```

setWindowDisplayScale Scales the image before displaying

## Parameters

<i>scale</i>	Window scale size
--------------	-------------------

## 7.10.4.30 show()

```
void cf::Window2D::show ( ) const
```

show Show image, on first call it may require additional time to display content correctly (in those cases use wait↵Key(1000) )

#### 7.10.4.31 waitKey()

```
unsigned char cf::Window2D::waitKey (
    int delay = 0 ) const
```

waitKey Block access until key input on window

##### Parameters

<i>delay</i>	Value > 0 -> wait till key input on window or 'delay'ms else wait till user input
--------------	---

##### Returns

#### 7.10.4.32 waitMouseInput()

```
void cf::Window2D::waitMouseInput (
    float & x,
    float & y )
```

waitMouseInput Blocks until mouse input has been given

##### Parameters

<i>x</i>	X-Window position
<i>y</i>	Y-Window position

### 7.10.5 Member Data Documentation

#### 7.10.5.1 m\_FristShowCall

```
bool cf::Window2D::m_FristShowCall = true [mutable], [protected]
```

#### 7.10.5.2 m\_Image

```
cv::Mat cf::Window2D::m_Image [protected]
```

#### 7.10.5.3 m\_IntervalChanged

```
bool cf::Window2D::m_IntervalChanged = false [protected]
```

#### 7.10.5.4 m\_IntervalX

```
cf::Interval cf::Window2D::m_IntervalX [protected]
```

## 7.10.5.5 m\_IntervalY

```
cf::Interval cf::Window2D::m_IntervalY [protected]
```

## 7.10.5.6 m\_InvertYAxis

```
bool cf::Window2D::m_InvertYAxis [protected]
```

## 7.10.5.7 m\_MouseCallBackStorage

```
float cf::Window2D::m_MouseCallBackStorage[2] [protected]
```

## 7.10.5.8 m\_WindowName

```
const char* cf::Window2D::m_WindowName [protected]
```

## 7.10.5.9 m\_WindowScale

```
float cf::Window2D::m_WindowScale [protected]
```

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

- include/window2D.h

## 7.11 cf::Window3D Struct Reference

The [Window3D](#) struct is the default class for accessing 3D content, creating more than 1 instance results in undefined behavior.

```
#include <window3D.h>
```

## Public Types

- enum [CameraType](#) {  
[CameraType::NONE](#), [CameraType::ROTATION](#), [CameraType::FREE\\_MOVEMENT](#), [CameraType::STATIC\\_X\\_AXIS](#),  
[CameraType::STATIC\\_Y\\_AXIS](#), [CameraType::STATIC\\_Z\\_AXIS](#) }

*The CameraType enum providing access to camera types, default: '[CameraType::ROTATION](#)'.*

## Public Member Functions

- [Window3D](#) (int \*argc, char \*\*argv, int width=800, int height=600, const char \*title="chaos and fractals")
- virtual [~Window3D](#) ()
- void [clear](#) (const [Color](#) &color=[Color::BLACK](#))
- virtual void [draw](#) ()=0  
*draw Draw function, this has to be implemented*
- virtual void [handleKeyboardInput](#) (unsigned char key, int x, int y)  
*handleKeyboardInput Access key input by simple override this function*
- int [startDrawing](#) ()  
*startDrawing Start drawing, this function only returns afer 'ESC'-key press*
- int [getWindowWidth](#) () const
- int [getWindowHeight](#) () const
- void [setCamera](#) ([CameraType](#) type, glm::vec3 lookAt=glm::vec3(0, 0, 0), float distance=10.f)  
*setCamera Set or change current camera type*
- void [drawAxis](#) (float length=10.f) const  
*drawAxis Draw x-,y- and z-axis*
- void [forceDisplay](#) () const  
*forceDisplay Displays all content, it may be used for displaying the current process of the draw function*
- void [drawCylinder](#) (const glm::vec3 &drawingDirection, const glm::vec3 &position, float diameter=1.f, const [Color](#) &color=[Color::WHITE](#)) const  
*drawCylinder Draws a solid cylinder*
- void [drawCylinder](#) (const glm::vec4 &drawingDirection, const glm::vec3 &position, float diameter=1.f, const [Color](#) &color=[Color::WHITE](#)) const  
*Type adjusted version of [Window3D::drawCylinder](#).*
- void [drawCylinder](#) (const glm::vec3 &drawingDirection, const glm::vec4 &position, float diameter=1.f, const [Color](#) &color=[Color::WHITE](#)) const  
*Type adjusted version of [Window3D::drawCylinder](#).*
- void [drawCylinder](#) (const glm::vec4 &drawingDirection, const glm::vec4 &position, float diameter=1.f, const [Color](#) &color=[Color::WHITE](#)) const  
*Type adjusted version of [Window3D::drawCylinder](#).*
- void [setMaxFPS](#) (float maxFPS=0.f)  
*setMaxFPS Set maximum frames per second*
- void [enableLighting](#) ()  
*enableLighting Enable lightning (Default: lightning is enabled)*
- void [disableLighting](#) ()  
*disableLighting Disable lightning (Default: lightning is enabled)*

## Static Public Member Functions

- static void [printWindowUsage](#) ()  
*printWindowUsage Print camera usage to console*

## Protected Member Functions

- void [\\_AdjustCamera](#) ()

## Protected Attributes

- float [m\\_DistAdjustment](#) = 1.f
- float [m\\_AngleAdjustment](#) = 1.f
- float [m\\_CameraAdjustment](#) = 1.f
- glm::vec3 [m\\_LookAt](#) = glm::vec3(0.f, 0.f, 0.f)
- float [m\\_LookAtDistance](#) = 10.f
- float [m\\_RotationAngle\\_Y](#) = 0.f
- float [m\\_RotationAngle\\_X](#) = 0.f
- [CameraType m\\_CameraType](#) = [Window3D::CameraType::ROTATION](#)
- glm::vec3 [m\\_FreeCamera\\_position](#) = glm::vec3(0.f, 0.f, 0.f)  
[CameraType::FREE\\_MOVEMENT](#) specific member variables.
- glm::vec3 [m\\_FreeCamera\\_UpVector](#) = glm::vec3(0.f, 1.f, 0.f)
- glm::vec3 [m\\_FreeCamera\\_LookDirection](#) = glm::vec3(0.f, 0.f, 1.f)

## Friends

- void [\\_KeyboardCallbackFunction](#) (unsigned char key, int x, int y)
- void [\\_DrawingFunction](#) ()

### 7.11.1 Detailed Description

The [Window3D](#) struct is the default class for accessing 3D content, creating more than 1 instance results in undefined behavior.

### 7.11.2 Member Enumeration Documentation

#### 7.11.2.1 CameraType

```
enum cf::Window3D::CameraType [strong]
```

The CameraType enum providing access to camera types, default: '[CameraType::ROTATION](#)'.

#### Enumerator

NONE	
ROTATION	
FREE_MOVEMENT	
STATIC_X_AXIS	
STATIC_Y_AXIS	
STATIC_Z_AXIS	

### 7.11.3 Constructor & Destructor Documentation

#### 7.11.3.1 Window3D()

```
cf::Window3D::Window3D (  
    int * argc,
```

```
char ** argv,
int width = 800,
int height = 600,
const char * title = "chaos and fractals" )
```

#### 7.11.3.2 ~Window3D()

```
virtual cf::Window3D::~~Window3D ( ) [virtual]
```

### 7.11.4 Member Function Documentation

#### 7.11.4.1 \_AdjustCamera()

```
void cf::Window3D::_AdjustCamera ( ) [protected]
```

#### 7.11.4.2 clear()

```
void cf::Window3D::clear (
    const Color & color = Color::BLACK )
```

#### 7.11.4.3 disableLighting()

```
void cf::Window3D::disableLighting ( ) [inline]
```

disableLighting Disable lightning (Default: lightning is enabled)

#### 7.11.4.4 draw()

```
virtual void cf::Window3D::draw ( ) [pure virtual]
```

draw Draw function, this has to be implemented

#### 7.11.4.5 drawAxis()

```
void cf::Window3D::drawAxis (
    float length = 10.f ) const
```

drawAxis Draw x-,y- and z-axis

#### Parameters

<i>length</i>	Axis length
---------------	-------------

## 7.11.4.6 drawCylinder() [1/4]

```
void cf::Window3D::drawCylinder (
    const glm::vec3 & drawingDirection,
    const glm::vec3 & position,
    float diameter = 1.f,
    const Color & color = Color::WHITE ) const
```

drawCylinder Draws a solid cylinder

## Parameters

<i>drawingDirection</i>	Cylinder direction
<i>position</i>	Start position
<i>diameter</i>	Cylinder diameter
<i>color</i>	Cylinder color

## 7.11.4.7 drawCylinder() [2/4]

```
void cf::Window3D::drawCylinder (
    const glm::vec4 & drawingDirection,
    const glm::vec3 & position,
    float diameter = 1.f,
    const Color & color = Color::WHITE ) const
```

Type adjusted version of [Window3D::drawCylinder](#).

## 7.11.4.8 drawCylinder() [3/4]

```
void cf::Window3D::drawCylinder (
    const glm::vec3 & drawingDirection,
    const glm::vec4 & position,
    float diameter = 1.f,
    const Color & color = Color::WHITE ) const
```

Type adjusted version of [Window3D::drawCylinder](#).

## 7.11.4.9 drawCylinder() [4/4]

```
void cf::Window3D::drawCylinder (
    const glm::vec4 & drawingDirection,
    const glm::vec4 & position,
    float diameter = 1.f,
    const Color & color = Color::WHITE ) const
```

Type adjusted version of [Window3D::drawCylinder](#).

#### 7.11.4.10 enableLighting()

```
void cf::Window3D::enableLighting ( ) [inline]
```

enableLighting Enable lightning (Default: lightning is enabled)

#### 7.11.4.11 forceDisplay()

```
void cf::Window3D::forceDisplay ( ) const
```

forceDisplay Displays all content, it may be used for displaying the current process of the draw function

#### 7.11.4.12 getWindowHeight()

```
int cf::Window3D::getWindowHeight ( ) const
```

#### 7.11.4.13 getWindowWidth()

```
int cf::Window3D::getWindowWidth ( ) const
```

#### 7.11.4.14 handleKeyboardInput()

```
virtual void cf::Window3D::handleKeyboardInput (
    unsigned char key,
    int x,
    int y ) [virtual]
```

handleKeyboardInput Access key input by simple override this function

##### Parameters

<i>key</i>	Key pressed
<i>x</i>	Mouse-x-position of the key press event
<i>y</i>	Mouse-y-position of the key press event

#### 7.11.4.15 printWindowUsage()

```
static void cf::Window3D::printWindowUsage ( ) [static]
```

printWindowUsage Print camera usage to console

#### 7.11.4.16 setCamera()

```
void cf::Window3D::setCamera (
    CameraType type,
```



```
glm::vec3 lookAt = glm::vec3(0, 0, 0),
float distance = 10.f )
```

setCamera Set or change current camera type

#### Parameters

<i>type</i>	Camera type
<i>lookAt</i>	
<i>distance</i>	

#### 7.11.4.17 setMaxFPS()

```
void cf::Window3D::setMaxFPS (
    float maxFPS = 0.f )
```

setMaxFPS Set maximum frames per second

#### Parameters

<i>maxFPS</i>	values > 0 indicates capped fps, value of 0 indicates "only draw after key-input", 0 is default
---------------	---

#### 7.11.4.18 startDrawing()

```
int cf::Window3D::startDrawing ( )
```

startDrawing Start drawing, this function only returns afer 'ESC'-key press

#### Returns

### 7.11.5 Friends And Related Function Documentation

#### 7.11.5.1 \_DrawingFunction

```
void _DrawingFunction ( ) [friend]
```

#### 7.11.5.2 \_KeyboardCallbackFunction

```
void _KeyboardCallbackFunction (
    unsigned char key,
    int x,
    int y ) [friend]
```

## 7.11.6 Member Data Documentation

### 7.11.6.1 m\_AngleAdjustment

`float cf::Window3D::m_AngleAdjustment = 1.f [protected]`

### 7.11.6.2 m\_CameraAdjustment

`float cf::Window3D::m_CameraAdjustment = 1.f [protected]`

### 7.11.6.3 m\_CameraType

`CameraType cf::Window3D::m_CameraType = Window3D::CameraType::ROTATION [protected]`

### 7.11.6.4 m\_DistAdjustment

`float cf::Window3D::m_DistAdjustment = 1.f [protected]`

### 7.11.6.5 m\_FreeCamera\_LookDirection

`glm::vec3 cf::Window3D::m_FreeCamera_LookDirection = glm::vec3(0.f, 0.f, 1.f) [protected]`

### 7.11.6.6 m\_FreeCamera\_position

`glm::vec3 cf::Window3D::m_FreeCamera_position = glm::vec3(0.f, 0.f, 0.f) [protected]`

[CameraType::FREE\\_MOVEMENT](#) specific member variables.

### 7.11.6.7 m\_FreeCamera\_UpVector

`glm::vec3 cf::Window3D::m_FreeCamera_UpVector = glm::vec3(0.f, 1.f, 0.f) [protected]`

### 7.11.6.8 m\_LookAt

`glm::vec3 cf::Window3D::m_LookAt = glm::vec3(0.f, 0.f, 0.f) [protected]`

### 7.11.6.9 m\_LookAtDistance

`float cf::Window3D::m_LookAtDistance = 10.f [protected]`

## 7.11.6.10 m\_RotationAngle\_X

```
float cf::Window3D::m_RotationAngle_X = 0.f [protected]
```

## 7.11.6.11 m\_RotationAngle\_Y

```
float cf::Window3D::m_RotationAngle_Y = 0.f [protected]
```

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

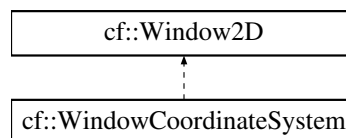
- include/window3D.h

## 7.12 cf::WindowCoordinateSystem Struct Reference

The [WindowCoordinateSystem](#) struct Default class for images and raster operations.

```
#include <windowCoordinateSystem.hpp>
```

Inheritance diagram for cf::WindowCoordinateSystem:



## Public Types

- enum [LineType](#)

*The LineType enum Special line type used by one function of 'drawLine'.*

## Public Member Functions

- [WindowCoordinateSystem](#) (int width, const [cf::Interval](#) &range\_x, const [cf::Interval](#) &range\_y, const char \*windowName="Computer Geometry", const [cf::Color](#) &startColor=[cf::Color::WHITE](#))  
*WindowCoordinateSystem Constructor.*
- virtual [~WindowCoordinateSystem](#) ()=default
- void [setInterval](#) (const [cf::Interval](#) &range\_x, const [cf::Interval](#) &range\_y, int width)  
*setInterval Set new interval*
- void [drawPoint](#) (const [cf::Point](#) &pos, const [cf::Color](#) &color=[cf::Color::BLACK](#))  
*drawPoint Draws a cross-shaped point*
- void [drawLine](#) (const [cf::Point](#) &p1, const [cf::Point](#) &p2, const [cf::Color](#) &color=[cf::Color::BLACK](#), [cf::Window2D::LineType](#) type=[cf::Window2D::LineType::DEFAULT](#), int lineWidth=1)  
*drawLine Draw a simple line of width 1*
- void [drawLinearEquation](#) (const [cf::Point](#) &pointVector, const glm::vec3 &drawingDirection, const [cf::Color](#) &color=[cf::Color::BLACK](#), [cf::Window2D::LineType](#) type=[cf::Window2D::LineType::DEFAULT](#), int lineWidth=1)  
*drawLinearEquation Draws a line from a point on line and direction vector*

- void [drawLinearEquation](#) (float a, float b, float c, const [cf::Color](#) &color=[cf::Color::BLACK](#), [cf::Window2D::LineType](#) type=[cf::Window2D::LineType::DEFAULT](#), int lineWidth=1)  
*drawLinearEquation Draw a line from a linear equation:  $ax + by + c = 0$*
- void [drawLinearEquation](#) (const [glm::vec3](#) &vec, const [cf::Color](#) &color=[cf::Color::BLACK](#), [cf::Window2D::LineType](#) type=[cf::Window2D::LineType::DEFAULT](#), int lineWidth=1)  
*drawLinearEquation Draw line from linear equation:  $ax + by + c = 0$ , where a b and c are part of coefficient vector*
- void [drawLinearEquation](#) (float slope, float yIntercept, const [cf::Color](#) &color=[cf::Color::BLACK](#), [cf::Window2D::LineType](#) type=[cf::Window2D::LineType::DEFAULT](#), int lineWidth=1)  
*drawLinearEquation Draw line from standard format  $y = m \cdot x + t$*
- void [drawCircle](#) (const [cf::Point](#) &center, float radius, const [cf::Color](#) &color=[cf::Color::BLACK](#), int lineWidth=1)  
*drawCircle Draws a circle with interval radius*
- float [convert\\_pixelLength\\_to\\_intervalLength](#) (float pixelLength) const  
*convert\_pixelLength\_to\_intervalLength Converts length from pixel to interval*
- float [convert\\_intervalLength\\_to\\_pixelLength](#) (float intervalLength) const  
*convert\_intervalLength\_to\_pixelLength Converts length from interval to pixel*
- void [drawCriclePart](#) (const [cf::Point](#) &center, float radius, float startAngle, float endAngle, const [cf::Color](#) &color=[cf::Color::BLACK](#), int lineWidth=1)  
*drawCriclePart Draw a partition of a circle*

## Additional Inherited Members

### 7.12.1 Detailed Description

The [WindowCoordinateSystem](#) struct Default class for images and raster operations.

### 7.12.2 Member Enumeration Documentation

#### 7.12.2.1 LineType

```
enum cf::Window2D::LineType [strong]
```

The LineType enum Special line type used by one function of 'drawLine'.

### 7.12.3 Constructor & Destructor Documentation

#### 7.12.3.1 WindowCoordinateSystem()

```
cf::WindowCoordinateSystem::WindowCoordinateSystem (
    int width,
    const cf::Interval & range_x,
    const cf::Interval & range_y,
    const char * windowName = "Computer Geometry",
    const cf::Color & startColor = cf::Color::WHITE ) [inline]
```

[WindowCoordinateSystem](#) Constructor.

## Parameters

<i>range</i> ↔ _x	Interval in x direction
<i>range</i> ↔ _y	Interval in y direction
<i>width</i>	Image width in pixel (hight will be determind automatically)

## 7.12.3.2 ~WindowCoordinateSystem()

```
virtual cf::WindowCoordinateSystem::~~WindowCoordinateSystem ( ) [virtual], [default]
```

## 7.12.4 Member Function Documentation

## 7.12.4.1 convert\_intervalLength\_to\_pixelLength()

```
float cf::WindowCoordinateSystem::convert_intervalLength_to_pixelLength (
    float intervalLength ) const [inline]
```

convert\_intervalLength\_to\_pixelLength Converts length from interval to pixel

## Parameters

<i>intervalLength</i>	
-----------------------	--

## Returns

## 7.12.4.2 convert\_pixelLength\_to\_intervalLength()

```
float cf::WindowCoordinateSystem::convert_pixelLength_to_intervalLength (
    float pixelLength ) const [inline]
```

convert\_pixelLength\_to\_intervalLength Converts length from pixel to interval

## Parameters

<i>pixelLength</i>	
--------------------	--

## Returns

#### 7.12.4.3 drawCircle()

```
void cf::WindowCoordinateSystem::drawCircle (
    const cf::Point & center,
    float radius,
    const cf::Color & color = cf::Color::BLACK,
    int lineWidth = 1 ) [inline]
```

drawCircle Draws a circle with interval radius

##### Parameters

<i>center</i>	Circle center
<i>radius</i>	Circle radius
<i>color</i>	Circle color
<i>lineWidth</i>	Width of the line, Note: only available on default line type

#### 7.12.4.4 drawCriclePart()

```
void cf::WindowCoordinateSystem::drawCriclePart (
    const cf::Point & center,
    float radius,
    float startAngle,
    float endAngle,
    const cf::Color & color = cf::Color::BLACK,
    int lineWidth = 1 ) [inline]
```

drawCriclePart Draw a partition of a circle

##### Parameters

<i>center</i>	Circle center
<i>radius</i>	Circle radius (in intervall length)
<i>startAngle</i>	Starting angle for circle (0° -> positive x direction, 90° -> positive y direction)
<i>endAngle</i>	End angle for circle (0° -> positive x-axis, 90° -> positive y-axis)
<i>color</i>	Circle color
<i>lineWidth</i>	Line width of the circle

#### 7.12.4.5 drawLine()

```
void cf::WindowCoordinateSystem::drawLine (
    const cf::Point & p1,
    const cf::Point & p2,
    const cf::Color & color = cf::Color::BLACK,
    cf::Window2D::LineType type = cf::Window2D::LineType::DEFAULT,
    int lineWidth = 1 ) [inline]
```

drawLine Draw a simple line of width 1

## Parameters

<i>p1</i>	First point
<i>p2</i>	Second point
<i>color</i>	Line color
<i>type</i>	Line type
<i>lineWidth</i>	Width of the line, Note: only available on default line type

## 7.12.4.6 drawLinearEquation() [1/4]

```
void cf::WindowCoordinateSystem::drawLinearEquation (
    const cf::Point & pointVector,
    const glm::vec3 & drawingDirection,
    const cf::Color & color = cf::Color::BLACK,
    cf::Window2D::LineType type = cf::Window2D::LineType::DEFAULT,
    int lineWidth = 1 ) [inline]
```

drawLinearEquation Draws a line from a point on line and direction vector

## Parameters

<i>pointVector</i>	Point on the line
<i>drawingDirection</i>	Line direction
<i>color</i>	Line color
<i>type</i>	Change line type to dot/dash/dot-dash
<i>lineWidth</i>	Width of the line, Note: only available on default line type

## 7.12.4.7 drawLinearEquation() [2/4]

```
void cf::WindowCoordinateSystem::drawLinearEquation (
    float a,
    float b,
    float c,
    const cf::Color & color = cf::Color::BLACK,
    cf::Window2D::LineType type = cf::Window2D::LineType::DEFAULT,
    int lineWidth = 1 ) [inline]
```

drawLinearEquation Draw a line from a linear equation:  $ax + by + c = 0$

## Parameters

<i>a</i>	Coefficient of x
<i>b</i>	Coefficient of y
<i>c</i>	Constant
<i>color</i>	Line color
<i>type</i>	Change line type to dot/dash/dot-dash
<i>lineWidth</i>	Width of the line, Note: only available on default line type

## 7.12.4.8 drawLinearEquation() [3/4]

```
void cf::WindowCoordinateSystem::drawLinearEquation (
    const glm::vec3 & vec,
    const cf::Color & color = cf::Color::BLACK,
    cf::Window2D::LineType type = cf::Window2D::LineType::DEFAULT,
    int lineWidth = 1 ) [inline]
```

drawLinearEquation Draw line from linear equation:  $ax + by + c = 0$ , where a b and c are part of coefficient vector

## Parameters

<i>vec</i>	Vector of coefficients a b and c
<i>color</i>	Line color
<i>type</i>	Change line type to dot/dash/dot-dash
<i>lineWidth</i>	Width of the line, Note: only available on default line type

## 7.12.4.9 drawLinearEquation() [4/4]

```
void cf::WindowCoordinateSystem::drawLinearEquation (
    float slope,
    float yIntercept,
    const cf::Color & color = cf::Color::BLACK,
    cf::Window2D::LineType type = cf::Window2D::LineType::DEFAULT,
    int lineWidth = 1 ) [inline]
```

drawLinearEquation Draw line from standard format  $y = m \cdot x + t$

## Parameters

<i>slope</i>	Slope m of equation $y = m \cdot x + t$
<i>yIntercept</i>	y-Intercept t of equation $y = m \cdot x + t$
<i>color</i>	Line color
<i>type</i>	Change line type to dot/dash/dot-dash
<i>lineWidth</i>	Width of the line, Note: only available on default line type

## 7.12.4.10 drawPoint()

```
void cf::WindowCoordinateSystem::drawPoint (
    const cf::Point & pos,
    const cf::Color & color = cf::Color::BLACK ) [inline]
```

drawPoint Draws a cross-shaped point

## Parameters

<i>pos</i>	Cross position
<i>color</i>	Cross color



## 7.12.4.11 setInterval()

```
void cf::WindowCoordinateSystem::setInterval (
    const cf::Interval & range_x,
    const cf::Interval & range_y,
    int width ) [inline]
```

setInterval Set new interval

## Parameters

<i>range</i> ↔ _x	Interval in x direction
<i>range</i> ↔ _y	Interval in y direction
<i>width</i>	Image width in pixel (hight will be determind automatically)

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

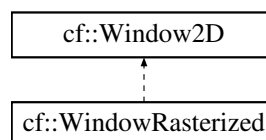
- include/[windowCoordinateSystem.hpp](#)

## 7.13 cf::WindowRasterized Struct Reference

The [WindowRasterized](#) struct Default struct for vectorized operations within a custom interval.

```
#include <windowRasterized.hpp>
```

Inheritance diagram for cf::WindowRasterized:



## Public Types

- enum [LineType](#)

*The LineType enum Special line type used by one function of 'drawLine'.*

## Public Member Functions

- [WindowRasterized](#) (int width=800, int height=600, const char \*windowName="Chaos and Fractals", const [cf::Color](#) &startColor={0, 0, 0})  
*WindowRasterized Constructor.*
- [WindowRasterized](#) (const char \*filePath)  
*WindowRasterized Load image from file path.*
- virtual [~WindowRasterized](#) ()=default

## Additional Inherited Members

### 7.13.1 Detailed Description

The [WindowRasterized](#) struct Default struct for vectorized operations within a custom interval.

### 7.13.2 Member Enumeration Documentation

#### 7.13.2.1 LineType

```
enum cf::Window2D::LineType [strong]
```

The LineType enum Special line type used by one function of 'drawLine'.

### 7.13.3 Constructor & Destructor Documentation

#### 7.13.3.1 WindowRasterized() [1/2]

```
cf::WindowRasterized::WindowRasterized (
    int width = 800,
    int height = 600,
    const char * windowName = "Chaos and Fractals",
    const cf::Color & startColor = {0, 0, 0} ) [inline]
```

[WindowRasterized](#) Constructor.

##### Parameters

<i>width</i>	Pixel width of the image
<i>height</i>	Pixel height of the image
<i>windowName</i>	Name of the window
<i>startColor</i>	Background color

#### 7.13.3.2 WindowRasterized() [2/2]

```
cf::WindowRasterized::WindowRasterized (
    const char * filePath ) [inline]
```

[WindowRasterized](#) Load image from file path.

##### Parameters

<i>filePath</i>	Path to file
-----------------	--------------

## 7.13.3.3 ~WindowRasterized()

```
virtual cf::WindowRasterized::~~WindowRasterized ( ) [virtual], [default]
```

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

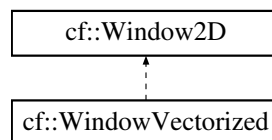
- include/windowRasterized.hpp

## 7.14 cf::WindowVectorized Struct Reference

The [WindowVectorized](#) struct Default class for images and raster operations.

```
#include <windowVercorized.hpp>
```

Inheritance diagram for cf::WindowVectorized:



## Public Types

- enum [LineType](#)  
The LineType enum Special line type used by one function of 'drawLine'.

## Public Member Functions

- [WindowVectorized](#) (int width, const [cf::Interval](#) &range\_x, const [cf::Interval](#) &range\_y, const char \*windowName="Chaos and Fractals", const [cf::Color](#) &startColor=[cf::Color::BLACK](#))  
*WindowVectorized Constructor.*
- [WindowVectorized](#) (const char \*filePath, int width, const [cf::Interval](#) &range\_x, const [cf::Interval](#) &range\_y)  
*WindowVectorized Image reading constructor.*
- virtual [~WindowVectorized](#) ()=default
- void [setInterval](#) (const [cf::Interval](#) &range\_x, const [cf::Interval](#) &range\_y, int width)  
*setInterval Set new interval*
- [cf::Point](#) [transformPoint\\_fromInterval\\_toImage](#) ([cf::Point](#) point)  
*transformPoint\_fromInterval\_toImage Transform point from interval position to pixel position*
- [cf::Point](#) [transformPoint\\_fromImage\\_toInterval](#) ([cf::Point](#) point)  
*transformPoint\_fromImage\_toInterval Transform point from pixel position to interval position*
- float [convert\\_pixelLength\\_to\\_intervalLength](#) (float pixelLength) const  
*convert\_pixelLength\_to\_intervalLength Converts length from pixel to interval*
- float [convert\\_intervalLength\\_to\\_pixelLength](#) (float intervalLength) const  
*convert\_intervalLength\_to\_pixelLength Converts length from interval to pixel*
- [cf::Color](#) [getColor\\_imageSpace](#) (int i, int j) const  
*getColor\_imageSpace Get color from image i/j position*
- void [setColor\\_imageSpace](#) (int i, int j, const [cf::Color](#) &color)  
*setColor\_imageSpace Set color from image i/j position*

## Additional Inherited Members

### 7.14.1 Detailed Description

The [WindowVectorized](#) struct Default class for images and raster operations.

### 7.14.2 Member Enumeration Documentation

#### 7.14.2.1 LineType

```
enum cf::Window2D::LineType [strong]
```

The LineType enum Special line type used by one function of 'drawLine'.

### 7.14.3 Constructor & Destructor Documentation

#### 7.14.3.1 WindowVectorized() [1/2]

```
cf::WindowVectorized::WindowVectorized (
    int width,
    const cf::Interval & range_x,
    const cf::Interval & range_y,
    const char * windowName = "Chaos and Fractals",
    const cf::Color & startColor = cf::Color::BLACK ) [inline]
```

[WindowVectorized](#) Constructor.

##### Parameters

<i>width</i>	Image width in pixel (hight will be determind automatically)
<i>range</i> <sub>↔</sub> <i>_x</i>	<a href="#">Interval</a> in x direction
<i>range</i> <sub>↔</sub> <i>_y</i>	<a href="#">Interval</a> in y direction

#### 7.14.3.2 WindowVectorized() [2/2]

```
cf::WindowVectorized::WindowVectorized (
    const char * filePath,
    int width,
    const cf::Interval & range_x,
    const cf::Interval & range_y ) [inline]
```

[WindowVectorized](#) Image reading constructoor.

##### Parameters

<i>filePath</i>	Path to image file
-----------------	--------------------

## Parameters

<i>width</i>	Image width, Note: height will be calculated based on ranges and width
<i>range</i> ↔ _x	Interval in x direction
<i>range</i> ↔ _y	Interval in y direction

## 7.14.3.3 ~WindowVectorized()

```
virtual cf::WindowVectorized::~~WindowVectorized ( ) [virtual], [default]
```

## 7.14.4 Member Function Documentation

## 7.14.4.1 convert\_intervalLength\_to\_pixelLength()

```
float cf::WindowVectorized::convert_intervalLength_to_pixelLength (
    float intervalLength ) const [inline]
```

convert\_intervalLength\_to\_pixelLength Converts length from interval to pixel

## Parameters

<i>intervalLength</i>	Length to be converted to pixel length
-----------------------	--

## Returns

## 7.14.4.2 convert\_pixelLength\_to\_intervalLength()

```
float cf::WindowVectorized::convert_pixelLength_to_intervalLength (
    float pixelLength ) const [inline]
```

convert\_pixelLength\_to\_intervalLength Converts length from pixel to interval

## Parameters

<i>pixelLength</i>	Length to be converted to the interval length
--------------------	---

## Returns

## 7.14.4.3 getColor\_imageSpace()

```
cf::Color cf::WindowVectorized::getColor_imageSpace (
    int i,
    int j ) const [inline]
```

getColor\_imageSpace Get color from image i/j position

## Parameters

<i>i</i>	I position
<i>j</i>	J position

## Returns

## 7.14.4.4 setColor\_imageSpace()

```
void cf::WindowVectorized::setColor_imageSpace (
    int i,
    int j,
    const cf::Color & color ) [inline]
```

setColor\_imageSpace Set color from image i/j position

## Parameters

<i>i</i>	I position
<i>j</i>	J position
<i>color</i>	Color to be set

## 7.14.4.5 setInterval()

```
void cf::WindowVectorized::setInterval (
    const cf::Interval & range_x,
    const cf::Interval & range_y,
    int width ) [inline]
```

setInterval Set new interval

## Parameters

<i>range<sub>x</sub></i>	Interval in x direction
<i>range<sub>y</sub></i>	Interval in y direction
<i>width</i>	Image width in pixel (hight will be determind automatically)

## 7.14.4.6 transformPoint\_fromImage\_toInterval()

```
cf::Point cf::WindowVectorized::transformPoint_fromImage_toInterval (
    cf::Point point ) [inline]
```

transformPoint\_fromImage\_toInterval Transform point from pixel position to interval position

## Parameters

<i>point</i>	Point to be transformed
--------------	-------------------------

## Returns

Transformed point

## 7.14.4.7 transformPoint\_fromInterval\_tolmage()

```
cf::Point cf::WindowVectorized::transformPoint_fromInterval_toImage (
    cf::Point point ) [inline]
```

transformPoint\_fromInterval\_tolmage Transform point from interval position to pixel position

## Parameters

<i>point</i>	Point to be transformed
--------------	-------------------------

## Returns

Transformed point

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

- include/[windowVercorized.hpp](#)





## Chapter 8

# File Documentation

### 8.1 include/computerGeometry.hpp File Reference

```
#include "windowCoordinateSystem.hpp"
#include "utils.h"
#include <sstream>
#include <fstream>
#include <string>
```

#### Classes

- struct [cf::Vec3< POINTVECTOR >](#)  
*The [Vec3](#) struct General class for vector operations.*
- struct [cf::Vec3< POINTVECTOR >](#)  
*The [Vec3](#) struct General class for vector operations.*

#### Namespaces

- [cf](#)

#### Typedefs

- typedef Vec3< true > [cf::PointVector](#)  
*PointVector Specialiaztion of general [Vec3](#).*
- typedef Vec3< false > [cf::DirectionVector](#)  
*DirectionVector Specialiaztion of general [Vec3](#), where component 'w' may not be written to.*

#### Functions

- template<bool b>  
std::ostream & [operator<<](#) (std::ostream &os, const [cf::Vec3< b >](#) &rhs)  
*operator<< Simple shift operator for output*

### 8.1.1 Function Documentation

#### 8.1.1.1 `operator<<()`

```
template<bool b>
std::ostream & operator<< (
    std::ostream & os,
    const cf::Vec3< b > & rhs )
```

`operator<<` Simple shift operator for output

##### Parameters

<code>os</code>	Outputstream, e.g. <code>std::cout</code>
<code>rhs</code>	<a href="#">cf::PointVector</a> or <a href="#">cf::DirectionVector</a>

##### Returns

## 8.2 `include/IFS.h` File Reference

```
#include "utils.h"
```

### Classes

- struct [cf::IteratedFunctionSystem](#)

*The [IteratedFunctionSystem](#) class lazy people (like myself) may use the IFS typedef.*

### Namespaces

- [cf](#)

### Typedefs

- typedef IteratedFunctionSystem [cf::IFS](#)

## 8.3 `include/LSystem.h` File Reference

```
#include <string>
#include <vector>
#include <memory>
#include <glm/glm.hpp>
#include "utils.h"
```

## Classes

- struct [cf::LindenmayerSystem](#)

The [LindenmayerSystem](#) class lazy people (like myself) may use the IFS typedef.

## Namespaces

- [cf](#)

## Typedefs

- typedef LindenmayerSystem [cf::LSystem](#)

## 8.4 include/ORB.h File Reference

```
#include "utils.h"
```

## Classes

- struct [cf::Orbit](#)

The [Orbit](#) class lazy people (like myself) may use the ORB typedef.

## Namespaces

- [cf](#)

## Typedefs

- typedef Orbit [cf::ORB](#)

## 8.5 include/utils.h File Reference

```
#include <string>
#include <vector>
#include <fstream>
#include <sstream>
#include <iostream>
#include <inttypes.h>
#include <glm/glm.hpp>
#include <glm/gtx/transform.hpp>
#include <glm/gtx/vector_angle.hpp>
#include <glm/gtx/rotate_vector.hpp>
```

## Classes

- struct [cf::Direction](#)  
The *Direction* struct for getting absolute directions from a current direction and a relative direction.
- struct [cf::Interval](#)  
The *Interval* struct provides functionality to translate values from one interval into another.
- struct [cf::Color](#)  
The *Color* struct offers a class for rgb access.
- struct [cf::Console](#)  
The *Console* struct offers utility functions for 'console'.

## Namespaces

- [cf](#)

## Functions

- `std::ostream & operator<< (std::ostream &of, const glm::vec2 &vec)`
- `std::ostream & operator<< (std::ostream &of, const glm::vec3 &vec)`
- `std::ostream & operator<< (std::ostream &of, const glm::vec4 &vec)`
- `std::ostream & operator<< (std::ostream &of, const glm::mat3x3 &mat)`
- `std::ostream & operator<< (std::ostream &of, const glm::mat4x4 &mat)`
- `void cf::\_removeWindowsSpecificCarriageReturn (std::string &str)`  
*\_removeWindowsSpecificCarriageReturn Removes 'carriage return' characters in strings ('carriage return' may be read from unix system by providing windows files)*
- `std::vector< Color > cf::readPaletteFromFile (const std::string &filePath)`  
*readPaletteFromFile*
- `std::string cf::readAntString (const std::string &filePath)`  
*readAntString*
- `template<typename _VectorType = glm::vec3>  
std::vector< _VectorType > cf::readDATFile (const std::string &filePath)`  
*readDATFile Reads a \*.dat file*
- `float cf::radian2degree (float radianValue)`  
*radian2degree Converts a radian value to a degree value*
- `float cf::degree2radian (float degreeValue)`  
*degree2radian Converts a degree value to a radian value*

### 8.5.1 Function Documentation

#### 8.5.1.1 `operator<<()` [1/5]

```
std::ostream& operator<< (
    std::ostream & of,
    const glm::vec2 & vec )
```

#### 8.5.1.2 `operator<<()` [2/5]

```
std::ostream& operator<< (
    std::ostream & of,
    const glm::vec3 & vec )
```

**8.5.1.3 operator<<()** [3/5]

```
std::ostream& operator<< (
    std::ostream & of,
    const glm::vec4 & vec )
```

**8.5.1.4 operator<<()** [4/5]

```
std::ostream& operator<< (
    std::ostream & of,
    const glm::mat3x3 & mat )
```

**8.5.1.5 operator<<()** [5/5]

```
std::ostream& operator<< (
    std::ostream & of,
    const glm::mat4x4 & mat )
```

**8.6 include/window2D.h File Reference**

```
#include <opencv2/opencv.hpp>
#include "utils.h"
```

**Classes**

- class [cf::Window2D](#)  
The [Window2D](#) struct offers advanced features used by *WindowRasterized/WindowVectorized*.
- struct [cf::Point](#)  
The [Point](#) struct is a simple class for position access on 2D images (imilar to *cv::Point*, but uses floats instead of integer)

**Namespaces**

- [cf](#)

**8.7 include/window3D.h File Reference**

```
#include <GL/freeglut.h>
#include <functional>
#include <vector>
#include <string>
#include "utils.h"
```

## Classes

- struct [cf::Window3D](#)

*The [Window3D](#) struct is the default class for accessing 3D content, creating more than 1 instance results in undefined behavior.*

## Namespaces

- [cf](#)

## 8.8 include/windowCoordinateSystem.hpp File Reference

```
#include "window2D.h"
```

## Classes

- struct [cf::WindowCoordinateSystem](#)

*The [WindowCoordinateSystem](#) struct Default class for images and raster operations.*

## Namespaces

- [cf](#)

## 8.9 include/windowRasterized.hpp File Reference

```
#include "window2D.h"
```

## Classes

- struct [cf::WindowRasterized](#)

*The [WindowRasterized](#) struct Default struct for vectorized operations within a custom interval.*

## Namespaces

- [cf](#)

## 8.10 include/windowVercorized.hpp File Reference

```
#include "window2D.h"
```

## Classes

- struct [cf::WindowVectorized](#)

*The [WindowVectorized](#) struct Default class for images and raster operations.*

## Namespaces

- [cf](#)

## 8.11 README.md File Reference





# Index

- \_AdjustCamera
    - cf::Window3D, 56
  - \_DrawingFunction
    - cf::Window3D, 59
  - \_KeyboardCallbackFunction
    - cf::Window3D, 59
  - \_convertFromNewInterval
    - cf::Window2D, 45
  - \_convertToNewInterval
    - cf::Window2D, 45
  - \_correctYValue
    - cf::Window2D, 45
  - \_removeWindowsSpecificCarriageReturn
    - cf, 13
  - \_window2foreground
    - cf::Window2D, 45
- ~Window2D
  - cf::Window2D, 45
- ~Window3D
  - cf::Window3D, 56
- ~WindowCoordinateSystem
  - cf::WindowCoordinateSystem, 63
- ~WindowRasterized
  - cf::WindowRasterized, 68
- ~WindowVectorized
  - cf::WindowVectorized, 71
- AbsoluteDirection
  - cf::Direction, 23
- b
  - cf::Color, 20
- BLACK
  - cf::Color, 20
- BLUE
  - cf::Color, 20
- CYAN
  - cf::Color, 20
- CameraType
  - cf::Window3D, 55
- cf, 11
  - \_removeWindowsSpecificCarriageReturn, 13
  - degree2radian, 13
  - DirectionVector, 12
  - IFS, 12
  - LSystem, 12
  - ORB, 12
  - PointVector, 12
  - radian2degree, 13
  - readAntString, 14
  - readDATFile, 14
  - readPaletteFromFile, 14
- cf::Color, 17
  - b, 20
  - BLACK, 20
  - BLUE, 20
  - CYAN, 20
  - Color, 18
  - g, 20
  - GREEN, 20
  - GREY, 20
  - invert, 18
  - MAGENTA, 20
  - ORANGE, 20
  - operator!=, 18
  - operator<<, 19
  - operator\*, 18, 19
  - operator\*=: 18
  - operator+, 18
  - operator+=, 18
  - operator-, 19
  - operator-=, 19
  - operator/, 19
  - operator/=, 19
  - operator==, 19
  - PINK, 21
  - r, 21
  - RED, 21
  - WHITE, 21
  - YELLOW, 21
- cf::Console, 21
  - clearConsole, 22
  - readFloat, 22
  - readInt, 22
  - readString, 22
  - waitKey, 22
- cf::Direction, 23
  - AbsoluteDirection, 23
  - getNextiDirection, 24
  - RelativeDirection, 24
  - toString, 24
- cf::Interval, 25
  - Interval, 25
  - max, 26
  - min, 26
  - operator<<, 26
  - translateIntervalPostion, 25
- cf::IteratedFunctionSystem, 26

- getAllTransformation, 27
- getName, 27
- getNumTransformations, 27
- getRangeX, 27
- getRangeY, 27
- getTransformation, 27
- read, 27
- cf::LindenmayerSystem, 28
  - clearWindowEachTime, 28
  - getAdjustmentAngel, 28
  - getAllProductions, 28
  - getAxiom, 28
  - getName, 28
  - getNumProductions, 29
  - getProduction, 29
  - getRangeX, 29
  - getRangeY, 29
  - getScale, 29
  - getStartAngle, 29
  - read, 29
- cf::Orbit, 30
  - getAllFactors, 30
  - getAllStartingPoints, 30
  - getName, 30
  - getNumFactors, 30
  - getNumStartingPoints, 30
  - getRangeX, 30
  - getRangeY, 31
  - read, 31
- cf::Point, 31
  - operator cv::Point, 32
  - operator!=, 32
  - operator\*, 32, 33
  - operator\*==, 32
  - operator+, 32
  - operator+=, 33
  - operator-, 33
  - operator-=, 33
  - operator/, 33
  - operator/==, 33
  - operator==, 33
  - Point, 32
  - x, 34
  - y, 34
- cf::Vec3
  - getW, 36
  - getX, 36
  - getY, 36
  - isPointVector, 37
  - normalize, 37
  - operator cf::Point, 37
  - operator cf::Vec3< false >, 37
  - operator const glm::vec3 &, 37
  - operator glm::vec3, 37
  - operator<<>, 41
  - operator\*, 38, 41
  - operator\*==, 39
  - operator+, 39
  - operator+=, 39
  - operator-, 39
  - operator-=, 39
  - operator=, 39, 40
  - operator%, 38
  - operator%==, 38
  - operator[], 40
  - setW, 40
  - setX, 41
  - setY, 41
  - Vec3, 36
  - Vec3<!POINTVECTOR >, 42
- cf::Vec3< POINTVECTOR >, 34
- cf::Window2D, 42
  - \_convertFromNewInterval, 45
  - \_convertToNewInterval, 45
  - \_correctYValue, 45
  - \_window2foreground, 45
  - ~Window2D, 45
  - clear, 45
  - drawAxis, 45
  - drawCircle, 46
  - drawCriclePart, 46
  - drawLine, 47
  - drawRectangle, 47
  - drawSpecializedLine, 47
  - flippHorizontal, 48
  - flippVertical, 48
  - floodFill, 48
  - getColor, 48
  - getHeight, 48
  - getImage, 49
  - getIntervalX, 49
  - getIntervalY, 49
  - getInvertYAxis, 49
  - getWidth, 49
  - getWindowDisplayScale, 50
  - LineType, 44
  - m\_FristShowCall, 52
  - m\_Image, 52
  - m\_IntervalChanged, 52
  - m\_IntervalX, 52
  - m\_IntervalY, 52
  - m\_InvertYAxis, 53
  - m\_MouseCallBackStorage, 53
  - m\_WindowName, 53
  - m\_WindowScale, 53
  - resetInterval, 50
  - resize, 50
  - savelImage, 50
  - setColor, 50
  - setInvertYAxis, 51
  - setNewInterval, 51
  - setWindowDisplayScale, 51
  - show, 51
  - waitKey, 51
  - waitMouseInput, 52
  - Window2D, 45

- cf::Window3D, 53
  - \_AdjustCamera, 56
  - \_DrawingFunction, 59
  - \_KeyboardCallbackFunction, 59
  - ~Window3D, 56
  - CameraType, 55
  - clear, 56
  - disableLighting, 56
  - draw, 56
  - drawAxis, 56
  - drawCylinder, 56, 57
  - enableLighting, 57
  - forceDisplay, 58
  - getWindowHeight, 58
  - getWindowWidth, 58
  - handleKeyboardInput, 58
  - m\_AngleAdjustment, 60
  - m\_CameraAdjustment, 60
  - m\_CameraType, 60
  - m\_DistAdjustment, 60
  - m\_FreeCamera\_LookDirection, 60
  - m\_FreeCamera\_UpVector, 60
  - m\_FreeCamera\_position, 60
  - m\_LookAt, 60
  - m\_LookAtDistance, 60
  - m\_RotationAngle\_X, 60
  - m\_RotationAngle\_Y, 61
  - printWindowUsage, 58
  - setCamera, 58
  - setMaxFPS, 59
  - startDrawing, 59
  - Window3D, 55
- cf::WindowCoordinateSystem, 61
  - ~WindowCoordinateSystem, 63
  - convert\_intervalLength\_to\_pixelLength, 63
  - convert\_pixelLength\_to\_intervalLength, 63
  - drawCircle, 63
  - drawCriclePart, 64
  - drawLine, 64
  - drawLinearEquation, 65, 66
  - drawPoint, 66
  - LineType, 62
  - setInterval, 66
  - WindowCoordinateSystem, 62
- cf::WindowRasterized, 67
  - ~WindowRasterized, 68
  - LineType, 68
  - WindowRasterized, 68
- cf::WindowVectorized, 69
  - ~WindowVectorized, 71
  - convert\_intervalLength\_to\_pixelLength, 71
  - convert\_pixelLength\_to\_intervalLength, 71
  - getColor\_imageSpace, 71
  - LineType, 70
  - setColor\_imageSpace, 72
  - setInterval, 72
  - transformPoint\_fromImage\_toInterval, 72
  - transformPoint\_fromInterval\_toImage, 73
  - WindowVectorized, 70
- clear
  - cf::Window2D, 45
  - cf::Window3D, 56
- clearConsole
  - cf::Console, 22
- clearWindowEachTime
  - cf::LindenmayerSystem, 28
- Color
  - cf::Color, 18
- computerGeometry.hpp
  - operator<<, 76
- convert\_intervalLength\_to\_pixelLength
  - cf::WindowCoordinateSystem, 63
  - cf::WindowVectorized, 71
- convert\_pixelLength\_to\_intervalLength
  - cf::WindowCoordinateSystem, 63
  - cf::WindowVectorized, 71
- degree2radian
  - cf, 13
- DirectionVector
  - cf, 12
- disableLighting
  - cf::Window3D, 56
- draw
  - cf::Window3D, 56
- drawAxis
  - cf::Window2D, 45
  - cf::Window3D, 56
- drawCircle
  - cf::Window2D, 46
  - cf::WindowCoordinateSystem, 63
- drawCriclePart
  - cf::Window2D, 46
  - cf::WindowCoordinateSystem, 64
- drawCylinder
  - cf::Window3D, 56, 57
- drawLine
  - cf::Window2D, 47
  - cf::WindowCoordinateSystem, 64
- drawLinearEquation
  - cf::WindowCoordinateSystem, 65, 66
- drawPoint
  - cf::WindowCoordinateSystem, 66
- drawRectangle
  - cf::Window2D, 47
- drawSpecializedLine
  - cf::Window2D, 47
- enableLighting
  - cf::Window3D, 57
- flippHorizontal
  - cf::Window2D, 48
- flippVertical
  - cf::Window2D, 48
- floodFill
  - cf::Window2D, 48

- forceDisplay
  - cf::Window3D, 58
- g
  - cf::Color, 20
- GREEN
  - cf::Color, 20
- GREY
  - cf::Color, 20
- getAdjustmentAngel
  - cf::LindenmayerSystem, 28
- getAllFactors
  - cf::Orbit, 30
- getAllProductions
  - cf::LindenmayerSystem, 28
- getAllStartingPoints
  - cf::Orbit, 30
- getAllTransformation
  - cf::IteratedFunctionSystem, 27
- getAxiom
  - cf::LindenmayerSystem, 28
- getColor
  - cf::Window2D, 48
- getColor\_imageSpace
  - cf::WindowVectorized, 71
- getHeight
  - cf::Window2D, 48
- getImage
  - cf::Window2D, 49
- getIntervalX
  - cf::Window2D, 49
- getIntervalY
  - cf::Window2D, 49
- getInvertYAxis
  - cf::Window2D, 49
- getName
  - cf::IteratedFunctionSystem, 27
  - cf::LindenmayerSystem, 28
  - cf::Orbit, 30
- getNextDirection
  - cf::Direction, 24
- getNumFactors
  - cf::Orbit, 30
- getNumProductions
  - cf::LindenmayerSystem, 29
- getNumStartingPoints
  - cf::Orbit, 30
- getNumTransformations
  - cf::IteratedFunctionSystem, 27
- getProduction
  - cf::LindenmayerSystem, 29
- getRangeX
  - cf::IteratedFunctionSystem, 27
  - cf::LindenmayerSystem, 29
  - cf::Orbit, 30
- getRangeY
  - cf::IteratedFunctionSystem, 27
  - cf::LindenmayerSystem, 29
  - cf::Orbit, 31
- getScale
  - cf::LindenmayerSystem, 29
- getStartAngle
  - cf::LindenmayerSystem, 29
- getTransformation
  - cf::IteratedFunctionSystem, 27
- getWidth
  - cf::Window2D, 49
- getWindowDisplayScale
  - cf::Window2D, 50
- getWindowHeight
  - cf::Window3D, 58
- getWindowWidth
  - cf::Window3D, 58
- getW
  - cf::Vec3, 36
- getX
  - cf::Vec3, 36
- getY
  - cf::Vec3, 36
- handleKeyboardInput
  - cf::Window3D, 58
- IFS
  - cf, 12
- include/IFS.h, 76
- include/LSystem.h, 76
- include/ORB.h, 77
- include/computerGeometry.hpp, 75
- include/utils.h, 77
- include/window2D.h, 79
- include/window3D.h, 79
- include/windowCoordinateSystem.hpp, 80
- include/windowRasterized.hpp, 80
- include/windowVercorized.hpp, 80
- Interval
  - cf::Interval, 25
- invert
  - cf::Color, 18
- isPointVector
  - cf::Vec3, 37
- LSystem
  - cf, 12
- LineType
  - cf::Window2D, 44
  - cf::WindowCoordinateSystem, 62
  - cf::WindowRasterized, 68
  - cf::WindowVectorized, 70
- m\_AngleAdjustment
  - cf::Window3D, 60
- m\_CameraAdjustment
  - cf::Window3D, 60
- m\_CameraType
  - cf::Window3D, 60
- m\_DistAdjustment
  - cf::Window3D, 60

m\_FreeCamera\_LookDirection  
     cf::Window3D, 60  
 m\_FreeCamera\_UpVector  
     cf::Window3D, 60  
 m\_FreeCamera\_position  
     cf::Window3D, 60  
 m\_FristShowCall  
     cf::Window2D, 52  
 m\_Image  
     cf::Window2D, 52  
 m\_IntervalChanged  
     cf::Window2D, 52  
 m\_IntervalX  
     cf::Window2D, 52  
 m\_IntervalY  
     cf::Window2D, 52  
 m\_InvertYAxis  
     cf::Window2D, 53  
 m\_LookAt  
     cf::Window3D, 60  
 m\_LookAtDistance  
     cf::Window3D, 60  
 m\_MouseCallBackStorage  
     cf::Window2D, 53  
 m\_RotationAngle\_X  
     cf::Window3D, 60  
 m\_RotationAngle\_Y  
     cf::Window3D, 61  
 m\_WindowName  
     cf::Window2D, 53  
 m\_WindowScale  
     cf::Window2D, 53  
 MAGENTA  
     cf::Color, 20  
 max  
     cf::Interval, 26  
 min  
     cf::Interval, 26  
 normalize  
     cf::Vec3, 37  
 ORANGE  
     cf::Color, 20  
 ORB  
     cf, 12  
 operator cf::Point  
     cf::Vec3, 37  
 operator cf::Vec3< false >  
     cf::Vec3, 37  
 operator const glm::vec3 &  
     cf::Vec3, 37  
 operator cv::Point  
     cf::Point, 32  
 operator glm::vec3  
     cf::Vec3, 37  
 operator!=  
     cf::Color, 18  
     cf::Point, 32  
 operator<<  
     cf::Color, 19  
     cf::Interval, 26  
     computerGeometry.hpp, 76  
     utils.h, 78, 79  
 operator<<()  
     cf::Vec3, 41  
 operator\*  
     cf::Color, 18, 19  
     cf::Point, 32, 33  
     cf::Vec3, 38, 41  
 operator\*=  
     cf::Color, 18  
     cf::Point, 32  
     cf::Vec3, 39  
 operator+  
     cf::Color, 18  
     cf::Point, 32  
     cf::Vec3, 39  
 operator+=  
     cf::Color, 18  
     cf::Point, 33  
     cf::Vec3, 39  
 operator-  
     cf::Color, 19  
     cf::Point, 33  
     cf::Vec3, 39  
 operator-=  
     cf::Color, 19  
     cf::Point, 33  
     cf::Vec3, 39  
 operator/  
     cf::Color, 19  
     cf::Point, 33  
 operator/=  
     cf::Color, 19  
     cf::Point, 33  
 operator=  
     cf::Vec3, 39, 40  
 operator==  
     cf::Color, 19  
     cf::Point, 33  
 operator%  
     cf::Vec3, 38  
 operator%=  
     cf::Vec3, 38  
 operator[]  
     cf::Vec3, 40  
 PINK  
     cf::Color, 21  
 Point  
     cf::Point, 32  
 PointVector  
     cf, 12  
 printWindowUsage  
     cf::Window3D, 58  
 r

- cf::Color, 21
- README.md, 81
- RED
  - cf::Color, 21
- radian2degree
  - cf, 13
- read
  - cf::IteratedFunctionSystem, 27
  - cf::LindenmayerSystem, 29
  - cf::Orbit, 31
- readAntString
  - cf, 14
- readDATFile
  - cf, 14
- readFloat
  - cf::Console, 22
- readInt
  - cf::Console, 22
- readPaletteFromFile
  - cf, 14
- readString
  - cf::Console, 22
- RelativeDirection
  - cf::Direction, 24
- resetInterval
  - cf::Window2D, 50
- resize
  - cf::Window2D, 50
- savelImage
  - cf::Window2D, 50
- setCamera
  - cf::Window3D, 58
- setColor
  - cf::Window2D, 50
- setColor\_imageSpace
  - cf::WindowVectorized, 72
- setInterval
  - cf::WindowCoordinateSystem, 66
  - cf::WindowVectorized, 72
- setInvertYAxis
  - cf::Window2D, 51
- setMaxFPS
  - cf::Window3D, 59
- setNewInterval
  - cf::Window2D, 51
- setWindowDisplayScale
  - cf::Window2D, 51
- setW
  - cf::Vec3, 40
- setX
  - cf::Vec3, 41
- setY
  - cf::Vec3, 41
- show
  - cf::Window2D, 51
- startDrawing
  - cf::Window3D, 59
- toString
  - cf::Direction, 24
- transformPoint\_fromImage\_toInterval
  - cf::WindowVectorized, 72
- transformPoint\_fromInterval\_toImage
  - cf::WindowVectorized, 73
- translateIntervalPostion
  - cf::Interval, 25
- utils.h
  - operator<<, 78, 79
- Vec3
  - cf::Vec3, 36
- Vec3<!POINTVECTOR >
  - cf::Vec3, 42
- WHITE
  - cf::Color, 21
- waitKey
  - cf::Console, 22
  - cf::Window2D, 51
- waitMouseInput
  - cf::Window2D, 52
- Window2D
  - cf::Window2D, 45
- Window3D
  - cf::Window3D, 55
- WindowCoordinateSystem
  - cf::WindowCoordinateSystem, 62
- WindowRasterized
  - cf::WindowRasterized, 68
- WindowVectorized
  - cf::WindowVectorized, 70
- x
  - cf::Point, 34
- y
  - cf::Point, 34
- YELLOW
  - cf::Color, 21