

Voxel World

0.0.3

Generated by Doxygen 1.8.14



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	Camera Class Reference . . . . .	5
3.1.1	Constructor & Destructor Documentation . . . . .	6
3.1.1.1	Camera() [1/2] . . . . .	6
3.1.1.2	Camera() [2/2] . . . . .	6
3.1.1.3	~Camera() . . . . .	6
3.1.2	Member Function Documentation . . . . .	6
3.1.2.1	getForward() . . . . .	7
3.1.2.2	getFov() . . . . .	7
3.1.2.3	getPerspectiveMatrix() . . . . .	7
3.1.2.4	getPosition() . . . . .	7
3.1.2.5	getSide() . . . . .	7
3.1.2.6	getUp() . . . . .	7
3.1.2.7	getViewMatrix() . . . . .	7
3.1.2.8	incline() . . . . .	7
3.1.2.9	move() . . . . .	8
3.1.2.10	resetCamera() . . . . .	8
3.1.2.11	resetView() . . . . .	8

3.1.2.12	<a href="#">setLookDirection()</a>	8
3.1.2.13	<a href="#">setPosition()</a>	8
3.1.2.14	<a href="#">turnH()</a>	8
3.1.2.15	<a href="#">turnV()</a>	8
3.1.3	<a href="#">Member Data Documentation</a>	9
3.1.3.1	<a href="#">forward</a>	9
3.1.3.2	<a href="#">fov</a>	9
3.1.3.3	<a href="#">height</a>	9
3.1.3.4	<a href="#">orig_forward</a>	9
3.1.3.5	<a href="#">orig_position</a>	9
3.1.3.6	<a href="#">orig_side</a>	9
3.1.3.7	<a href="#">orig_up</a>	10
3.1.3.8	<a href="#">position</a>	10
3.1.3.9	<a href="#">side</a>	10
3.1.3.10	<a href="#">up</a>	10
3.1.3.11	<a href="#">width</a>	10
3.1.3.12	<a href="#">zFar</a>	10
3.1.3.13	<a href="#">zNear</a>	11
3.2	<a href="#">Chunk Class Reference</a>	11
3.2.1	<a href="#">Constructor &amp; Destructor Documentation</a>	11
3.2.1.1	<a href="#">Chunk() [1/3]</a>	12
3.2.1.2	<a href="#">Chunk() [2/3]</a>	12
3.2.1.3	<a href="#">Chunk() [3/3]</a>	12
3.2.1.4	<a href="#">~Chunk()</a>	12
3.2.2	<a href="#">Member Function Documentation</a>	12
3.2.2.1	<a href="#">check_neighbour()</a>	12
3.2.2.2	<a href="#">create_cubes()</a>	12
3.2.2.3	<a href="#">operator()()</a>	13
3.2.2.4	<a href="#">send_render_data()</a>	13
3.2.2.5	<a href="#">update()</a>	13

3.2.2.6	<a href="#">update_visible_cubes()</a>	13
3.2.3	<a href="#">Member Data Documentation</a>	13
3.2.3.1	<a href="#">chunk_cubes</a>	13
3.2.3.2	<a href="#">cubes_info</a>	13
3.2.3.3	<a href="#">position</a>	14
3.2.3.4	<a href="#">render_data</a>	14
3.2.3.5	<a href="#">world</a>	14
3.3	<a href="#">Chunk_Holder Class Reference</a>	14
3.3.1	<a href="#">Constructor &amp; Destructor Documentation</a>	14
3.3.1.1	<a href="#">Chunk_Holder() [1/2]</a>	15
3.3.1.2	<a href="#">Chunk_Holder() [2/2]</a>	15
3.3.1.3	<a href="#">~Chunk_Holder()</a>	15
3.3.2	<a href="#">Member Function Documentation</a>	15
3.3.2.1	<a href="#">operator()()</a>	15
3.3.2.2	<a href="#">shift()</a>	15
3.3.3	<a href="#">Member Data Documentation</a>	15
3.3.3.1	<a href="#">chunkBox</a>	16
3.3.3.2	<a href="#">world</a>	16
3.4	<a href="#">cirArray&lt; T &gt; Class Template Reference</a>	16
3.4.1	<a href="#">Detailed Description</a>	16
3.4.2	<a href="#">Constructor &amp; Destructor Documentation</a>	16
3.4.2.1	<a href="#">cirArray() [1/2]</a>	17
3.4.2.2	<a href="#">cirArray() [2/2]</a>	17
3.4.3	<a href="#">Member Function Documentation</a>	17
3.4.3.1	<a href="#">operator=()</a>	17
3.4.3.2	<a href="#">operator[]()</a>	17
3.4.3.3	<a href="#">shift()</a>	17
3.4.3.4	<a href="#">size()</a>	17
3.4.4	<a href="#">Member Data Documentation</a>	18
3.4.4.1	<a href="#">array</a>	18

3.4.4.2	start	18
3.5	Cube Class Reference	18
3.5.1	Detailed Description	19
3.5.2	Constructor & Destructor Documentation	19
3.5.2.1	Cube() [1/3]	19
3.5.2.2	Cube() [2/3]	19
3.5.2.3	Cube() [3/3]	19
3.5.2.4	~Cube()	19
3.5.3	Member Function Documentation	19
3.5.3.1	cleanup()	20
3.5.3.2	getMesh()	20
3.5.3.3	initialize()	20
3.5.3.4	update()	20
3.5.4	Member Data Documentation	20
3.5.4.1	cube_type	20
3.5.4.2	meshes	20
3.5.4.3	position	21
3.5.4.4	textures	21
3.5.4.5	transparent	21
3.6	Light Struct Reference	21
3.6.1	Member Data Documentation	21
3.6.1.1	color	21
3.6.1.2	intensity	22
3.6.1.3	position	22
3.7	Mesh Struct Reference	22
3.7.1	Constructor & Destructor Documentation	22
3.7.1.1	~Mesh()	22
3.7.2	Member Data Documentation	22
3.7.2.1	indices	23
3.7.2.2	normals	23

3.7.2.3	uv	23
3.7.2.4	vertices	23
3.8	Object_3D Class Reference	23
3.8.1	Constructor & Destructor Documentation	24
3.8.1.1	Object_3D()	24
3.8.2	Member Function Documentation	24
3.8.2.1	set_instance_data()	24
3.8.3	Member Data Documentation	24
3.8.3.1	layouts	25
3.8.3.2	mesh_indices	25
3.8.3.3	render_instances	25
3.8.3.4	types	25
3.8.3.5	VAO	25
3.8.3.6	VBOs	25
3.9	Renderer Class Reference	26
3.9.1	Constructor & Destructor Documentation	26
3.9.1.1	Renderer() [1/2]	27
3.9.1.2	Renderer() [2/2]	27
3.9.1.3	~Renderer()	27
3.9.2	Member Function Documentation	27
3.9.2.1	add_data()	27
3.9.2.2	add_Shader()	27
3.9.2.3	change_active_program()	27
3.9.2.4	clear()	28
3.9.2.5	find_shader()	28
3.9.2.6	make_program()	28
3.9.2.7	multi_render()	28
3.9.2.8	render()	28
3.9.2.9	set_camera()	28
3.9.2.10	update()	29

3.9.3	Member Data Documentation . . . . .	29
3.9.3.1	busy_queue . . . . .	29
3.9.3.2	cam . . . . .	29
3.9.3.3	current_program . . . . .	29
3.9.3.4	fragment_shaders . . . . .	29
3.9.3.5	render_queue . . . . .	29
3.9.3.6	shading_programs . . . . .	30
3.9.3.7	tessellation_shaders . . . . .	30
3.9.3.8	vertex_shaders . . . . .	30
3.10	Shader Class Reference . . . . .	30
3.10.1	Constructor & Destructor Documentation . . . . .	31
3.10.1.1	Shader() [1/2] . . . . .	31
3.10.1.2	Shader() [2/2] . . . . .	31
3.10.1.3	~Shader() . . . . .	31
3.10.2	Member Function Documentation . . . . .	31
3.10.2.1	clear() . . . . .	31
3.10.2.2	load_from_file() . . . . .	31
3.10.3	Member Data Documentation . . . . .	32
3.10.3.1	fileName . . . . .	32
3.10.3.2	shaderID . . . . .	32
3.10.3.3	type . . . . .	32
3.11	Texture Class Reference . . . . .	32
3.11.1	Constructor & Destructor Documentation . . . . .	33
3.11.1.1	Texture() . . . . .	33
3.11.1.2	~Texture() . . . . .	33
3.11.2	Member Function Documentation . . . . .	34
3.11.2.1	clear() . . . . .	34
3.11.2.2	load_to_GPU() . . . . .	34
3.11.3	Member Data Documentation . . . . .	34
3.11.3.1	height . . . . .	34



3.11.3.2	target	34
3.11.3.3	texture	34
3.11.3.4	textureID	35
3.11.3.5	width	35
3.12	World Class Reference	35
3.12.1	Constructor & Destructor Documentation	35
3.12.1.1	World()	36
3.12.1.2	~World()	36
3.12.2	Member Function Documentation	36
3.12.2.1	center_frame()	36
3.12.2.2	operator()()	36
3.12.2.3	send_render_data()	36
3.12.3	Member Data Documentation	36
3.12.3.1	h_radius	37
3.12.3.2	loaded_chunks	37
3.12.3.3	origin	37
3.12.3.4	v_radius	37
<b>4</b>	<b>File Documentation</b>	<b>39</b>
4.1	Cube.cpp File Reference	39
4.1.1	Detailed Description	39
4.1.2	Variable Documentation	39
4.1.2.1	obj_source_files	39
4.1.2.2	texture_source_files	40
4.2	Cube.hpp File Reference	40
4.2.1	Detailed Description	40
4.2.2	Enumeration Type Documentation	40
4.2.2.1	CubeID	40
4.2.3	Variable Documentation	41
4.2.3.1	cube_types	41
4.3	Helpers/cout-definitions.cpp File Reference	41

4.3.1	Detailed Description	41
4.3.2	Function Documentation	41
4.3.2.1	operator<<() [1/4]	42
4.3.2.2	operator<<() [2/4]	42
4.3.2.3	operator<<() [3/4]	42
4.3.2.4	operator<<() [4/4]	42
4.4	Helpers/cout-definitions.hpp File Reference	42
4.4.1	Detailed Description	43
4.4.2	Function Documentation	43
4.4.2.1	operator<<() [1/4]	43
4.4.2.2	operator<<() [2/4]	43
4.4.2.3	operator<<() [3/4]	43
4.4.2.4	operator<<() [4/4]	43
4.5	Helpers/system-libraries.hpp File Reference	44
4.5.1	Detailed Description	44
4.5.2	Macro Definition Documentation	44
4.5.2.1	GLEW_DYNAMIC	44
4.6	Helpers/tools.cpp File Reference	45
4.6.1	Detailed Description	45
4.6.2	Function Documentation	45
4.6.2.1	fade()	45
4.6.2.2	length()	46
4.6.2.3	noise_2D()	46
4.6.2.4	perlin_noise()	46
4.6.2.5	surflet()	46
4.6.2.6	vec_field_init()	46
4.6.3	Variable Documentation	46
4.6.3.1	mask	47
4.6.3.2	perm	47
4.6.3.3	size	47

4.6.3.4	<a href="#">vec_field_x</a>	47
4.6.3.5	<a href="#">vec_field_y</a>	47
4.7	<a href="#">Helpers/tools.hpp File Reference</a>	47
4.7.1	<a href="#">Detailed Description</a>	48
4.7.2	<a href="#">Function Documentation</a>	48
4.7.2.1	<a href="#">noise_2D()</a>	48
4.7.2.2	<a href="#">vec_field_init()</a>	48
4.8	<a href="#">Helpers/wavefront-loader.cpp File Reference</a>	48
4.8.1	<a href="#">Detailed Description</a>	49
4.8.2	<a href="#">Function Documentation</a>	49
4.8.2.1	<a href="#">load_obj()</a>	49
4.9	<a href="#">Helpers/wavefront-loader.hpp File Reference</a>	49
4.9.1	<a href="#">Detailed Description</a>	49
4.9.2	<a href="#">Function Documentation</a>	50
4.9.2.1	<a href="#">load_obj()</a>	50
4.10	<a href="#">main.cpp File Reference</a>	50
4.10.1	<a href="#">Detailed Description</a>	50
4.10.2	<a href="#">Typedef Documentation</a>	51
4.10.2.1	<a href="#">frame_duration</a>	51
4.10.2.2	<a href="#">world_duration</a>	51
4.10.3	<a href="#">Function Documentation</a>	51
4.10.3.1	<a href="#">main()</a>	51
4.10.3.2	<a href="#">render_loop()</a>	51
4.10.3.3	<a href="#">update_loop()</a>	51
4.11	<a href="#">Rendering/Camera/Camera.cpp File Reference</a>	51
4.11.1	<a href="#">Detailed Description</a>	52
4.12	<a href="#">Rendering/Camera/Camera.hpp File Reference</a>	52
4.12.1	<a href="#">Detailed Description</a>	52
4.13	<a href="#">Rendering/OpenGL-Wrappers.cpp File Reference</a>	52
4.13.1	<a href="#">Detailed Description</a>	53

4.13.2	Macro Definition Documentation	53
4.13.2.1	STB_IMAGE_IMPLEMENTATION	53
4.13.2.2	STB_IMAGE_WRITE_IMPLEMENTATION	53
4.13.3	Function Documentation	53
4.13.3.1	init_buffer()	54
4.13.3.2	verify_uniform_location()	54
4.13.4	Variable Documentation	54
4.13.4.1	Rendering_Handler	54
4.14	Rendering/OpenGL-Wrappers.hpp File Reference	54
4.14.1	Detailed Description	55
4.14.2	Enumeration Type Documentation	55
4.14.2.1	PROGRAM	55
4.14.3	Function Documentation	55
4.14.3.1	openGLerror()	55
4.14.4	Variable Documentation	56
4.14.4.1	Rendering_Handler	56
4.15	Rendering/Window-Management.cpp File Reference	56
4.15.1	Detailed Description	56
4.15.2	Macro Definition Documentation	56
4.15.2.1	CAM_SPEED	57
4.15.3	Function Documentation	57
4.15.3.1	callBackInit()	57
4.15.3.2	create_context()	57
4.15.3.3	createWindow()	57
4.15.3.4	cursor_pos_callback()	57
4.15.3.5	error_callback()	58
4.15.3.6	key_callback()	58
4.15.3.7	mouse_button_callback()	58
4.15.3.8	openGLerror()	58
4.16	Rendering/Window-Management.hpp File Reference	58

4.16.1 Detailed Description . . . . .	59
4.16.2 Function Documentation . . . . .	59
4.16.2.1 calculateFPS() . . . . .	59
4.16.2.2 callBackInit() . . . . .	59
4.16.2.3 create_context() . . . . .	59
4.16.2.4 createWindow() . . . . .	60
4.16.2.5 cursor_pos_callback() . . . . .	60
4.16.2.6 error_callback() . . . . .	60
4.16.2.7 key_callback() . . . . .	60
4.16.2.8 mouse_button_callback() . . . . .	60
4.17 World.cpp File Reference . . . . .	61
4.17.1 Detailed Description . . . . .	61
4.17.2 Macro Definition Documentation . . . . .	61
4.17.2.1 MESH . . . . .	61
4.17.3 Variable Documentation . . . . .	61
4.17.3.1 the_world . . . . .	61
4.18 World.hpp File Reference . . . . .	62
4.18.1 Detailed Description . . . . .	62
4.18.2 Macro Definition Documentation . . . . .	62
4.18.2.1 CHUNK_DIMS . . . . .	62
4.18.3 Variable Documentation . . . . .	62
4.18.3.1 the_world . . . . .	62
<b>Index</b>	<b>63</b>



# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">Camera</a>	5
<a href="#">Chunk</a>	11
<a href="#">Chunk_Holder</a>	14
<a href="#">cirArray&lt; T &gt;</a>	16
<a href="#">Cube</a>	18
<a href="#">Light</a>	21
<a href="#">Mesh</a>	22
<a href="#">Object_3D</a>	23
<a href="#">Renderer</a>	26
<a href="#">Shader</a>	30
<a href="#">Texture</a>	32
<a href="#">World</a>	35





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">Cube.cpp</a>	Definition of a generic cube object . . . . .	39
<a href="#">Cube.hpp</a>	Header for the definition of a generic cube object . . . . .	40
<a href="#">main.cpp</a>	Main file. Thread and global loop definitoins go here, as well as initialization . . . . .	50
<a href="#">World.cpp</a>	Definitions of all world related classes and methods . . . . .	61
<a href="#">World.hpp</a>	Header for the definition of a generic chunk object . . . . .	62
Helpers/ <a href="#">cout-definitions.cpp</a>	Implementation of the output functions for I/O debugging . . . . .	41
Helpers/ <a href="#">cout-definitions.hpp</a>	Header defining some output methods to print structures to the terminal . . . . .	42
Helpers/ <a href="#">system-libraries.hpp</a>	General header for system libraries . . . . .	44
Helpers/ <a href="#">tools.cpp</a>	Implementation of miscellaneous helping functions and structures . . . . .	45
Helpers/ <a href="#">tools.hpp</a>	Header for the definition of a generic chunk object . . . . .	47
Helpers/ <a href="#">wavefront-loader.cpp</a>	Defines methods needed to load wavefront (.obj) meshes . . . . .	48
Helpers/ <a href="#">wavefront-loader.hpp</a>	Header declaration of methods needed to load wavefront (.obj) meshes . . . . .	49
Rendering/ <a href="#">OpenGL-Wrappers.cpp</a>	Wrapper structures to abstract OpenGL function calls . . . . .	52
Rendering/ <a href="#">OpenGL-Wrappers.hpp</a>	Header to define variables, structure definitoins, include libraries... Shared among all rendering functions . . . . .	54
Rendering/ <a href="#">Window-Management.cpp</a>	File defining all relevant OpenGL and GLFW related functions needed to create an OpenGL context and GLFW window . . . . .	56
Rendering/ <a href="#">Window-Management.hpp</a>	Header for the context creation implementation. Exposes functions and defines needed included files . . . . .	58

Rendering/Camera/ <a href="#">Camera.cpp</a>	
Implementation of the camera header. Defines the behaviour for a generic camera . . . . .	51
Rendering/Camera/ <a href="#">Camera.hpp</a>	
Header declaration of functions and members for a generic camera class . . . . .	52

## Chapter 3

# Class Documentation

### 3.1 Camera Class Reference

```
#include <Camera.hpp>
```

#### Public Member Functions

- [Camera](#) (mat3 frame, vec3 pos, float w, float h)
- [Camera](#) ()
- [~Camera](#) ()
- mat4 [getViewMatrix](#) ()
- mat4 [getPerspectiveMatrix](#) ()
- void [setLookDirection](#) (vec3 v)
- void [move](#) (vec3 v)
- void [setPosition](#) (vec3 p)
- void [turnH](#) (float angle)
- void [turnV](#) (float angle)
- void [incline](#) (float angle)
- void [resetView](#) ()
- void [resetCamera](#) ()
- vec3 [getPosition](#) ()
- vec3 [getForward](#) ()
- vec3 [getUp](#) ()
- vec3 [getSide](#) ()
- float [getFov](#) ()

#### Private Attributes

- vec3 [forward](#)  
*vector indicating forward direction of the camera*
- vec3 [up](#)  
*vector indicating up direction of the camera*
- vec3 [side](#)  
*vector indicating side direction of the camera*
- vec3 [position](#)

- vector indicating the position of the camera*
- vec3 [orig\\_forward](#)
  - vector indicating forward direction of the camera*
- vec3 [orig\\_up](#)
  - vector indicating up direction of the camera*
- vec3 [orig\\_side](#)
  - vector indicating side direction of the camera*
- vec3 [orig\\_position](#)
  - vector indicating the position of the camera*
- float [fov](#)
  - field of view, dimensions, near clipping angle, far clipping angle*
- float [width](#)
- float [height](#)
- float [zNear](#)
- float [zFar](#)

### 3.1.1 Constructor & Destructor Documentation

#### 3.1.1.1 [Camera\(\)](#) [1/2]

```
Camera::Camera (
    mat3 frame,
    vec3 pos,
    float w,
    float h )
```

Parameter constructor

#### 3.1.1.2 [Camera\(\)](#) [2/2]

```
Camera::Camera ( )
```

Default constructor

#### 3.1.1.3 [~Camera\(\)](#)

```
Camera::~~Camera ( )
```

Destructor

### 3.1.2 Member Function Documentation

#### 3.1.2.1 getForward()

```
vec3 Camera::getForward ( )
```

Return the direction in which the camera is looking

#### 3.1.2.2 getFov()

```
float Camera::getFov ( )
```

Return the field of view of the camera

#### 3.1.2.3 getPerspectiveMatrix()

```
mat4 Camera::getPerspectiveMatrix ( )
```

Get the perspective matrix of the camera

#### 3.1.2.4 getPosition()

```
vec3 Camera::getPosition ( )
```

Return the global camera position

#### 3.1.2.5 getSide()

```
vec3 Camera::getSide ( )
```

Return the side direction of the camera

#### 3.1.2.6 getUp()

```
vec3 Camera::getUp ( )
```

Return the up direction of the camera

#### 3.1.2.7 getViewMatrix()

```
mat4 Camera::getViewMatrix ( )
```

Get the view matrix of the camera

#### 3.1.2.8 incline()

```
void Camera::incline (
    float angle )
```

Rotate the camera around its forward direction

### 3.1.2.9 move()

```
void Camera::move (
    vec3 v )
```

Move the camera by an offset  $v$

### 3.1.2.10 resetCamera()

```
void Camera::resetCamera ( )
```

Hard reset all camera values to defaults

### 3.1.2.11 resetView()

```
void Camera::resetView ( )
```

Orient the camera to it's default looking direction and orientation

### 3.1.2.12 setLookDirection()

```
void Camera::setLookDirection (
    vec3 v )
```

Orient the camera so that it looks in the direction of  $v$

### 3.1.2.13 setPosition()

```
void Camera::setPosition (
    vec3 p )
```

Place the camera at specified position  $p$

### 3.1.2.14 turnH()

```
void Camera::turnH (
    float angle )
```

Rotate the camera around it's *up direction*

### 3.1.2.15 turnV()

```
void Camera::turnV (
    float angle )
```

Rotate the camera around it's *side direction*

### 3.1.3 Member Data Documentation

#### 3.1.3.1 forward

```
vec3 Camera::forward [private]
```

vector indicating forward direction of the camera

#### 3.1.3.2 fov

```
float Camera::fov [private]
```

field of view, dimensions, near clipping angle, far clipping angle

#### 3.1.3.3 height

```
float Camera::height [private]
```

#### 3.1.3.4 orig\_forward

```
vec3 Camera::orig_forward [private]
```

vector indicating forward direction of the camera

#### 3.1.3.5 orig\_position

```
vec3 Camera::orig_position [private]
```

vector indicating the position of the camera

#### 3.1.3.6 orig\_side

```
vec3 Camera::orig_side [private]
```

vector indicating side direction of the camera

#### 3.1.3.7 orig\_up

```
vec3 Camera::orig_up [private]
```

vector indicating up direction of the camera

#### 3.1.3.8 position

```
vec3 Camera::position [private]
```

vector indicating the position of the camera

#### 3.1.3.9 side

```
vec3 Camera::side [private]
```

vector indicating side direction of the camera

#### 3.1.3.10 up

```
vec3 Camera::up [private]
```

vector indicating up direction of the camera

#### 3.1.3.11 width

```
float Camera::width [private]
```

#### 3.1.3.12 zFar

```
float Camera::zFar [private]
```



## 3.1.3.13 zNear

```
float Camera::zNear [private]
```

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

- Rendering/Camera/[Camera.hpp](#)
- Rendering/Camera/[Camera.cpp](#)

## 3.2 Chunk Class Reference

```
#include <World.hpp>
```

### Public Member Functions

- [Cube](#) \* [operator\(\)](#) (int, int, int)
- [Chunk](#) ()
- [Chunk](#) (vec3)
- [Chunk](#) (vec3, [World](#) \*)
- [~Chunk](#) ()
- void [create\\_cubes](#) (vec3)
- void [update](#) ()
- void [send\\_render\\_data](#) ([Renderer](#) \*)

### Public Attributes

- vec3 [position](#)

### Private Member Functions

- void [update\\_visible\\_cubes](#) ()
- bool [check\\_neighbour](#) ([Cube](#) \*c, [Cube](#) \*n)

### Private Attributes

- [World](#) \* [world](#)
- [Cube](#) \* [chunk\\_cubes](#) [[CHUNK\\_DIMS](#) \*[CHUNK\\_DIMS](#) \*[CHUNK\\_DIMS](#)] = {}
- [Object\\_3D](#) \* [render\\_data](#)
- vector< vec4 > [cubes\\_info](#)

### 3.2.1 Constructor & Destructor Documentation

### 3.2.1.1 `Chunk()` [1/3]

```
Chunk::Chunk ( )
```

Default constructor for a chunk

### 3.2.1.2 `Chunk()` [2/3]

```
Chunk::Chunk (
    vec3 offset )
```

Constructor for a chunk (should not be used, for testing only)

### 3.2.1.3 `Chunk()` [3/3]

```
Chunk::Chunk (
    vec3 offset,
    World * w )
```

Creates a chunk in the world *w* at global position *p*

### 3.2.1.4 `~Chunk()`

```
Chunk::~~Chunk ( )
```

Class destructor

## 3.2.2 Member Function Documentation

### 3.2.2.1 `check_neighbour()`

```
bool Chunk::check_neighbour (
    Cube * c,
    Cube * n ) [inline], [private]
```

Check neighbour information

### 3.2.2.2 `create_cubes()`

```
void Chunk::create_cubes (
    vec3 offset )
```

Creates cubes based on the current position of the chunk. This basically overwrites all previous data with new values associated with the current chunk location.

### 3.2.2.3 operator()

```
Cube * Chunk::operator() (
    int x,
    int y,
    int z )
```

() operator overloading, used to fetch cube values in the chunk and the world

### 3.2.2.4 send\_render\_data()

```
void Chunk::send_render_data (
    Renderer * handler ) [inline]
```

Send rendering information to the rendering handler

### 3.2.2.5 update()

```
void Chunk::update ( )
```

Update the current chunk

### 3.2.2.6 update\_visible\_cubes()

```
void Chunk::update_visible_cubes ( ) [private]
```

Set the cubes\_info array with the data of the currently visible cubes

## 3.2.3 Member Data Documentation

### 3.2.3.1 chunk\_cubes

```
Cube* Chunk::chunk_cubes [CHUNK_DIMS *CHUNK_DIMS *CHUNK_DIMS] = {} [private]
```

### 3.2.3.2 cubes\_info

```
vector<vec4> Chunk::cubes_info [private]
```

### 3.2.3.3 position

```
vec3 Chunk::position
```

### 3.2.3.4 render\_data

```
Object_3D* Chunk::render_data [private]
```

### 3.2.3.5 world

```
World* Chunk::world [private]
```

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

- [World.hpp](#)
- [World.cpp](#)

## 3.3 Chunk\_Holder Class Reference

```
#include <World.hpp>
```

### Public Member Functions

- [Chunk\\_Holder](#) ()
- [Chunk\\_Holder](#) (int, int, int, [World](#) \*)
- [~Chunk\\_Holder](#) ()
- [Chunk](#) \* [operator\(\)](#) (int, int, int)
- void [shift](#) (ivec3)

### Private Attributes

- [cirArray](#)< [cirArray](#)< [cirArray](#)< [Chunk](#) \* > > > [chunkBox](#)
- [World](#) \* [world](#)

### 3.3.1 Constructor & Destructor Documentation

### 3.3.1.1 Chunk\_Holder() [1/2]

```
Chunk_Holder::Chunk_Holder ( )
```

Default constructor

### 3.3.1.2 Chunk\_Holder() [2/2]

```
Chunk_Holder::Chunk_Holder (
    int x_dim,
    int y_dim,
    int z_dim,
    World * w )
```

Create a chunk holder of *x\_dim* , *y\_dim* , *z\_dim* dimensions in the world *w*

### 3.3.1.3 ~Chunk\_Holder()

```
Chunk_Holder::~~Chunk_Holder ( )
```

Class destructor

## 3.3.2 Member Function Documentation

### 3.3.2.1 operator()()

```
Chunk * Chunk_Holder::operator() (
    int x,
    int y,
    int z )
```

Overloaded () operator, used to fetch the chunk at local indices *x*, *y*, *z*

### 3.3.2.2 shift()

```
void Chunk_Holder::shift (
    ivec3 offset )
```

Shift the entire loaded box in the direction specified by *offset* This effectively moves the world into that direction  
Re-initialize *y* values as needed

## 3.3.3 Member Data Documentation

### 3.3.3.1 chunkBox

```
cirArray<cirArray<cirArray<Chunk*> > > Chunk_Holder::chunkBox [private]
```

### 3.3.3.2 world

```
World* Chunk_Holder::world [private]
```

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

- [World.hpp](#)
- [World.cpp](#)

## 3.4 cirArray< T > Class Template Reference

```
#include <tools.hpp>
```

### Public Member Functions

- [cirArray](#) ()
- [cirArray](#) (uint [size](#))
- void [shift](#) (int)
- T & [operator\[\]](#) (int)
- void [operator=](#) (T)
- uint [size](#) ()

### Private Attributes

- vector< T > [array](#)
- int [start](#)

### 3.4.1 Detailed Description

```
template<typename T>
class cirArray< T >
```

A generic circular array class, can be considered a ciircular vector

### 3.4.2 Constructor & Destructor Documentation

### 3.4.2.1 cirArray() [1/2]

```
template<typename T >
cirArray< T >::cirArray ( )
```

Default constructor of the class

### 3.4.2.2 cirArray() [2/2]

```
template<typename T >
cirArray< T >::cirArray (
    uint size )
```

Parametrized constructor of the class, creates a circular array of *size elements*

## 3.4.3 Member Function Documentation

### 3.4.3.1 operator=()

```
template<typename T>
void cirArray< T >::operator= (
    T )
```

### 3.4.3.2 operator[]()

```
template<typename T >
T & cirArray< T >::operator[] (
    int i )
```

[] operator, returns the element at index *i*

### 3.4.3.3 shift()

```
template<typename T >
void cirArray< T >::shift (
    int i )
```

shift the circular array by *i units*

### 3.4.3.4 size()

```
template<typename T >
uint cirArray< T >::size ( )
```

Returns the current size (number of elements) of the circular array

### 3.4.4 Member Data Documentation

#### 3.4.4.1 array

```
template<typename T>
vector<T> cirArray< T >::array [private]
```

#### 3.4.4.2 start

```
template<typename T>
int cirArray< T >::start [private]
```

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

- [Helpers/tools.hpp](#)

## 3.5 Cube Class Reference

```
#include <Cube.hpp>
```

### Public Member Functions

- void [update](#) (vec3 offset)
- [Cube](#) (vec3 p, [CubeID](#) type)
- [Cube](#) (vec3 p)
- [Cube](#) ()
- [~Cube](#) ()
- [Mesh](#) [getMesh](#) ()

### Static Public Member Functions

- static void [initialize](#) ()
- static void [cleanup](#) ()

### Public Attributes

- vec3 [position](#)  
*global position of the cube*
- [CubeID](#) [cube\\_type](#) = [DEFAULT](#)  
*unique ID representing the cube*
- bool [transparent](#) = false  
*is the block transparent?*



## Static Public Attributes

- static vector< [Mesh](#) \* > [meshes](#)  
*meshes of all cube subclasses*
- static vector< [Texture](#) \* > [textures](#)  
*textures of all cube subclasses*

### 3.5.1 Detailed Description

[Cube](#) class definition

### 3.5.2 Constructor & Destructor Documentation

#### 3.5.2.1 [Cube\(\)](#) [1/3]

```
Cube::Cube (
    vec3 p,
    CubeID type )
```

Parametrized constructor of the cube class *Type* refers to a cube ID

#### 3.5.2.2 [Cube\(\)](#) [2/3]

```
Cube::Cube (
    vec3 p )
```

Constructor for a cube, creates a default cube at position *p*

#### 3.5.2.3 [Cube\(\)](#) [3/3]

```
Cube::Cube ( )
```

Default constructor for a cube

#### 3.5.2.4 [~Cube\(\)](#)

```
Cube::~~Cube ( )
```

Destructor for a cube

### 3.5.3 Member Function Documentation

### 3.5.3.1 cleanup()

```
void Cube::cleanup ( ) [static]
```

Free memory used by the cube class

### 3.5.3.2 getMesh()

```
Mesh Cube::getMesh ( )
```

Get the value of the mesh of this cube

### 3.5.3.3 initialize()

```
void Cube::initialize ( ) [static]
```

Initialize the static memebrs of the [Cube](#) class

### 3.5.3.4 update()

```
void Cube::update (
    vec3 offset )
```

Update the position of the current cube and reset its values

## 3.5.4 Member Data Documentation

### 3.5.4.1 cube\_type

```
CubeID Cube::cube_type = DEFAULT
```

unique ID representing the cube

### 3.5.4.2 meshes

```
vector< Mesh * > Cube::meshes [static]
```

meshes of all cube subclasses

#### 3.5.4.3 position

```
vec3 Cube::position
```

global position of the cube

#### 3.5.4.4 textures

```
vector< Texture * > Cube::textures [static]
```

textures of all cube subclasses

#### 3.5.4.5 transparent

```
bool Cube::transparent = false
```

is the block transparent?

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

- [Cube.hpp](#)
- [Cube.cpp](#)

## 3.6 Light Struct Reference

```
#include <World.hpp>
```

### Public Attributes

- vec3 [position](#)
- vec4 [color](#)
- double [intensity](#)

### 3.6.1 Member Data Documentation

#### 3.6.1.1 color

```
vec4 Light::color
```

### 3.6.1.2 intensity

```
double Light::intensity
```

### 3.6.1.3 position

```
vec3 Light::position
```

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

- [World.hpp](#)

## 3.7 Mesh Struct Reference

```
#include <OpenGL-Wrappers.hpp>
```

### Public Member Functions

- [~Mesh\(\)](#)

### Public Attributes

- vector< vec3 > [vertices](#)  
*Vertex data.*
- vector< vec3 > [normals](#)  
*Normal data.*
- vector< uint > [indices](#)  
*Element data (sequence in which data will be read)*
- vector< vec2 > [uvs](#)

### 3.7.1 Constructor & Destructor Documentation

#### 3.7.1.1 ~Mesh()

```
Mesh::~~Mesh ( )
```

Class destructor

### 3.7.2 Member Data Documentation

### 3.7.2.1 indices

```
vector<uint> Mesh::indices
```

Element data (sequence in which data will be read)

### 3.7.2.2 normals

```
vector<vec3> Mesh::normals
```

Normal data.

### 3.7.2.3 uvs

```
vector<vec2> Mesh::uvs
```

[Texture](#) data for this geometry (the associated coordinates of the mesh)

### 3.7.2.4 vertices

```
vector<vec3> Mesh::vertices
```

Vertex data.

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

- [Rendering/OpenGL-Wrappers.hpp](#)
- [Rendering/OpenGL-Wrappers.cpp](#)

## 3.8 Object\_3D Class Reference

```
#include <OpenGL-Wrappers.hpp>
```

### Public Member Functions

- [Object\\_3D](#) ([Mesh](#) \*)
- [template<class T >](#)  
void [set\\_instance\\_data](#) ([Renderer](#) \*, [vector< T >](#))

## Public Attributes

- GLuint [VAO](#)  
*Vertex Array Object.*
- vector< GLuint > [VBOs](#)  
*array of VBO Ids*
- vector< GLuint > [types](#)  
*Array of VBO types.*
- uint [layouts](#)  
*The number of layouts to activate.*
- uint [render\\_instances](#)  
*Number of instances to render current object.*
- uint [mesh\\_indices](#)  
*Indices for index rendering, if any.*

## 3.8.1 Constructor & Destructor Documentation

### 3.8.1.1 Object\_3D()

```
Object_3D::Object_3D (
    Mesh * mesh )
```

Create a 3D rendereable object from a mesh

## 3.8.2 Member Function Documentation

### 3.8.2.1 set\_instance\_data()

```
template<class T >
void Object_3D::set_instance_data (
    Renderer * handler,
    vector< T > info )
```

Set the visual data for the current 3D object (SSBO data)

## 3.8.3 Member Data Documentation

### 3.8.3.1 layouts

```
uint Object_3D::layouts
```

The number of layouts to activate.

### 3.8.3.2 mesh\_indices

```
uint Object_3D::mesh_indices
```

Indices for index rendering, if any.

### 3.8.3.3 render\_instances

```
uint Object_3D::render_instances
```

Number of instances to render current object.

### 3.8.3.4 types

```
vector<GLuint> Object_3D::types
```

Array of VBO types.

### 3.8.3.5 VAO

```
GLuint Object_3D::VAO
```

Vertex Array Object.

### 3.8.3.6 VBOs

```
vector<GLuint> Object_3D::VBOs
```

array of VBO ids

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

- [Rendering/OpenGL-Wrappers.hpp](#)
- [Rendering/OpenGL-Wrappers.cpp](#)

## 3.9 Renderer Class Reference

```
#include <OpenGL-Wrappers.hpp>
```

### Public Member Functions

- [Renderer](#) ()
- [Renderer](#) (int width, int height)
- [~Renderer](#) ()
- [Shader](#) \* [find\\_shader](#) (string shader\_name)
- void [update](#) (GLFWwindow \*window)
- void [add\\_Shader](#) (string shader, GLuint type)
- void [make\\_program](#) (vector< uint > \*shaders)
- void [set\\_camera](#) ([Camera](#) \*new\_cam)
- void [multi\\_render](#) (GLuint VAO, vector< GLuint > \*VBOs, vector< GLuint > \*buffer\_types, GLuint layout↵\_num, GLuint index\_num, GLuint instances)
- void [change\\_active\\_program](#) (GLuint newProgram)
- void [add\\_data](#) ([Object\\_3D](#) \*)
- void [render](#) ()
- void [clear](#) ()

### Public Attributes

- mutex [busy\\_queue](#)  
*Lock to synchronize queue W/R.*
- [Camera](#) \* [cam](#)  
*Main (player) camera object.*
- GLuint [current\\_program](#)  
*Current shading program (program used to render)*

### Private Attributes

- vector< GLuint > [shading\\_programs](#)  
*Shading programs IDs.*
- vector< [Shader](#) > [vertex\\_shaders](#)  
*Vertex shader IDs.*
- vector< [Shader](#) > [fragment\\_shaders](#)  
*Fragment shader IDs.*
- vector< [Shader](#) > [tessellation\\_shaders](#)  
*Tessellation shader IDs.*
- vector< [Object\\_3D](#) \* > [render\\_queue](#)

#### 3.9.1 Constructor & Destructor Documentation



### 3.9.1.1 `Renderer()` [1/2]

```
Renderer::Renderer ( )
```

Default constructor for the [Renderer](#) Class

### 3.9.1.2 `Renderer()` [2/2]

```
Renderer::Renderer (
    int width,
    int height )
```

Contructor for the [Renderer](#) class. Creates a renderer object that handles all render calls. It's intended to be unique but has not been implemented as a singleton be weary!

### 3.9.1.3 `~Renderer()`

```
Renderer::~~Renderer ( )
```

Class destructor

## 3.9.2 Member Function Documentation

### 3.9.2.1 `add_data()`

```
void Renderer::add_data (
    Object\_3D * data )
```

Add a rendereable 3D object to the current render queue

### 3.9.2.2 `add_Shader()`

```
void Renderer::add_Shader (
    string shader,
    GLuint type )
```

Add a new shader to the set of all shaders

### 3.9.2.3 `change_active_program()`

```
void Renderer::change_active_program (
    GLuint newProgram )
```

#### 3.9.2.4 clear()

```
void Renderer::clear ( )
```

Clear all objects in the render queue

#### 3.9.2.5 find\_shader()

```
Shader * Renderer::find_shader (
    string shader_name )
```

Find a shader through a string

#### 3.9.2.6 make\_program()

```
void Renderer::make_program (
    vector< uint > * shaders )
```

#### 3.9.2.7 multi\_render()

```
void Renderer::multi_render (
    GLuint VAO,
    vector< GLuint > * VBOs,
    vector< GLuint > * buffer_types,
    GLuint layout_num,
    GLuint index_num,
    GLuint instances )
```

Function to render multiple instances of the same mesh index\_num is the number of indices in the mesh (for drawing elements) layout\_num is the number of layouts to enable (always 0 to layout\_num-1)

#### 3.9.2.8 render()

```
void Renderer::render ( )
```

Render all elements in the current render queue

#### 3.9.2.9 set\_camera()

```
void Renderer::set_camera (
    Camera * new_cam )
```

Initialize the main rendering camera

#### 3.9.2.10 update()

```
void Renderer::update (
    GLFWwindow * window )
```

Update general rendering values

### 3.9.3 Member Data Documentation

#### 3.9.3.1 busy\_queue

```
mutex Renderer::busy_queue
```

Lock to synchronize queue W/R.

#### 3.9.3.2 cam

```
Camera* Renderer::cam
```

Main (player) camera object.

#### 3.9.3.3 current\_program

```
GLuint Renderer::current_program
```

Current shading program (program used to render)

#### 3.9.3.4 fragment\_shaders

```
vector<Shader> Renderer::fragment_shaders [private]
```

Fragment shader IDs.

#### 3.9.3.5 render\_queue

```
vector<Object_3D*> Renderer::render_queue [private]
```

Queue of objects to render in the current frame

### 3.9.3.6 shading\_programs

```
vector<GLuint> Renderer::shading_programs [private]
```

Shading programs IDs.

### 3.9.3.7 tessellation\_shaders

```
vector<Shader> Renderer::tessellation_shaders [private]
```

Tessellation shader IDs.

### 3.9.3.8 vertex\_shaders

```
vector<Shader> Renderer::vertex_shaders [private]
```

Vertex shader IDs.

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

- Rendering/[OpenGL-Wrappers.hpp](#)
- Rendering/[OpenGL-Wrappers.cpp](#)

## 3.10 Shader Class Reference

```
#include <OpenGL-Wrappers.hpp>
```

### Public Member Functions

- [Shader](#) ()
- [Shader](#) (string file, GLenum [type](#))
- [~Shader](#) ()
- string [load\\_from\\_file](#) (string &)
- void [clear](#) ()

### Public Attributes

- string [fileName](#)  
*source file*
- GLuint [shaderID](#)  
*generated OpenGL shader ID*
- GLuint [type](#)  
*shader type*

### 3.10.1 Constructor & Destructor Documentation

#### 3.10.1.1 Shader() [1/2]

```
Shader::Shader ( )
```

Default Constructor

#### 3.10.1.2 Shader() [2/2]

```
Shader::Shader (
    string file,
    GLenum type )
```

Initialize the fields of a shader object using a glsl shader file

Parameters

<i>file</i>	the file path (relative or absolute) where the shader program is defined
<i>type</i>	the type of shader (e.g vertex,fragment, tessellation...)

#### 3.10.1.3 ~Shader()

```
Shader::~Shader ( )
```

Destructor of a shader struct

### 3.10.2 Member Function Documentation

#### 3.10.2.1 clear()

```
void Shader::clear ( )
```

Cleanup the shader OpenGL information

#### 3.10.2.2 load\_from\_file()

```
string Shader::load_from_file (
    string & filepath )
```

Copy a file into a a string

**Parameters**

<i>filepath</i>	path to the file
-----------------	------------------

**Returns**

A string that is the copy of the source file

### 3.10.3 Member Data Documentation

#### 3.10.3.1 fileName

```
string Shader::fileName
```

source file

#### 3.10.3.2 shaderID

```
GLuint Shader::shaderID
```

generated OpenGL shader ID

#### 3.10.3.3 type

```
GLuint Shader::type
```

shader type

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

- Rendering/[OpenGL-Wrappers.hpp](#)
- Rendering/[OpenGL-Wrappers.cpp](#)

## 3.11 Texture Class Reference

```
#include <OpenGL-Wrappers.hpp>
```

## Public Member Functions

- [Texture](#) (const char \*filename, GLuint [target](#)=GL\_TEXTURE\_2D)
- [~Texture](#) ()
- void [load\\_to\\_GPU](#) (GLuint)
- void [clear](#) ()

## Public Attributes

- GLuint [textureID](#)  
*OpenGL generated ID for the texture.*
- GLuint [target](#)  
*OpenGL target (Usually 2D texture or rectangle) check OpenGL doc.*
- string [texture](#)  
*Texture data.*
- int [width](#)  
*width of the texture*
- int [height](#)  
*height of the texture*

### 3.11.1 Constructor & Destructor Documentation

#### 3.11.1.1 Texture()

```
Texture::Texture (
    const char * filename,
    GLuint targ = GL_TEXTURE_2D )
```

Initialize the fields of a texture object using arrays

#### Parameters

<i>filename</i>	the filepath to the texture file
<i>targ</i>	the OpenGL texture target (e.g 2D, rectangle...)

#### Returns

Boolean value indicating whether an error occurred (true means no error)

#### 3.11.1.2 ~Texture()

```
Texture::~Texture ( )
```

Destructor of a texture struct

### 3.11.2 Member Function Documentation

#### 3.11.2.1 clear()

```
void Texture::clear ( )
```

Clear all OpenGL information of the texture object

#### 3.11.2.2 load\_to\_GPU()

```
void Texture::load_to_GPU (
    GLuint program )
```

### 3.11.3 Member Data Documentation

#### 3.11.3.1 height

```
int Texture::height
```

height of the texture

#### 3.11.3.2 target

```
GLuint Texture::target
```

OpenGL target (Usually 2D texture or rectangle) check OpenGL doc.

#### 3.11.3.3 texture

```
string Texture::texture
```

[Texture](#) data.



#### 3.11.3.4 textureID

```
GLuint Texture::textureID
```

OpenGL generated ID for the texture.

#### 3.11.3.5 width

```
int Texture::width
```

width of the texture

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

- Rendering/[OpenGL-Wrappers.hpp](#)
- Rendering/[OpenGL-Wrappers.cpp](#)

## 3.12 World Class Reference

```
#include <World.hpp>
```

### Public Member Functions

- [World](#) ()
- [~World](#) ()
- [Cube \\* operator\(\)](#) (int x, int y, int z)
- void [center\\_frame](#) (ivec3 offset)
- void [send\\_render\\_data](#) ([Renderer](#) \*)

### Public Attributes

- int [h\\_radius](#) = 7
- int [v\\_radius](#) = 4
- ivec3 [origin](#) = ivec3(0)

### Private Attributes

- [Chunk\\_Holder](#) \* [loaded\\_chunks](#)

### 3.12.1 Constructor & Destructor Documentation

### 3.12.1.1 World()

```
World::World ( )
```

[World](#) default constructor

### 3.12.1.2 ~World()

```
World::~~World ( )
```

[World](#) class destructor

## 3.12.2 Member Function Documentation

### 3.12.2.1 center\_frame()

```
void World::center_frame (
    ivec3 position )
```

Center the frame around *position*

### 3.12.2.2 operator>()()

```
Cube * World::operator() (
    int x,
    int y,
    int z )
```

Overloaded () operator, used to get chunk pointers in the loaded chunks of the world through their global position

### 3.12.2.3 send\_render\_data()

```
void World::send_render_data (
    Renderer * handler )
```

Send all world render data to the handler

## 3.12.3 Member Data Documentation

### 3.12.3.1 h\_radius

```
int World::h_radius = 7
```

### 3.12.3.2 loaded\_chunks

```
Chunk_Holder* World::loaded_chunks [private]
```

### 3.12.3.3 origin

```
ivec3 World::origin = ivec3(0)
```

### 3.12.3.4 v\_radius

```
int World::v_radius = 4
```

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

- [World.hpp](#)
- [World.cpp](#)



## Chapter 4

# File Documentation

### 4.1 Cube.cpp File Reference

Definition of a generic cube object.

```
#include "system-libraries.hpp"
#include "Cube.hpp"
#include "cout-definitions.hpp"
```

#### Variables

- `vector< string > texture_source_files = {"Assets/Textures/white_cube.png"}`
- `vector< string > obj_source_files = {"Assets/Objs/cube.obj"}`

#### 4.1.1 Detailed Description

Definition of a generic cube object.

#### Author

Camilo Talero

Version: 0.0.3

#### 4.1.2 Variable Documentation

##### 4.1.2.1 obj\_source\_files

```
vector<string> obj_source_files = {"Assets/Objs/cube.obj"}
```

#### 4.1.2.2 texture\_source\_files

```
vector<string> texture_source_files = {"Assets/Textures/white_cube.png"}
```

Global texture and mesh source file strings

## 4.2 Cube.hpp File Reference

Header for the definition of a generic cube object.

```
#include <string>
#include "OpenGL-Wrappers.hpp"
#include "wavefront-loader.hpp"
```

### Classes

- class [Cube](#)

### Enumerations

- enum [CubeID](#) { [DEFAULT](#) =0 }

### Variables

- const uint [cube\\_types](#) = 1

#### 4.2.1 Detailed Description

Header for the definition of a generic cube object.

#### Author

Camilo Talero

Version: 0.0.3

#### 4.2.2 Enumeration Type Documentation

##### 4.2.2.1 CubeID

```
enum CubeID
```

## Enumerator

DEFAULT	
---------	--

### 4.2.3 Variable Documentation

#### 4.2.3.1 cube\_types

```
const uint cube_types = 1
```

## 4.3 Helpers/cout-definitions.cpp File Reference

Implementation of the output functions for I/O debugging.

```
#include "cout-definitions.hpp"
```

### Functions

- ostream & [operator<<](#) (ostream &os, vec2 &v)
- ostream & [operator<<](#) (ostream &os, vec3 &v)
- ostream & [operator<<](#) (ostream &os, vec4 &v)
- ostream & [operator<<](#) (ostream &os, vector< float > &v)

#### 4.3.1 Detailed Description

Implementation of the output functions for I/O debugging.

## Author

Camilo Talero

Version: 0.0.3

#### 4.3.2 Function Documentation

#### 4.3.2.1 `operator<<()` [1/4]

```
ostream& operator<< (
    ostream & os,
    vec2 & v )
```

Print a vec2

#### 4.3.2.2 `operator<<()` [2/4]

```
ostream& operator<< (
    ostream & os,
    vec3 & v )
```

Print a vec3

#### 4.3.2.3 `operator<<()` [3/4]

```
ostream& operator<< (
    ostream & os,
    vec4 & v )
```

Print a vec4

#### 4.3.2.4 `operator<<()` [4/4]

```
ostream& operator<< (
    ostream & os,
    vector< float > & v )
```

Print a vector of floats

## 4.4 Helpers/cout-definitions.hpp File Reference

Header defining some output methods to print structures to the terminal.

```
#include "system-libraries.hpp"
```

### Functions

- ostream & [operator<<](#) (ostream &os, vec2 &v)
- ostream & [operator<<](#) (ostream &os, vec3 &v)
- ostream & [operator<<](#) (ostream &os, vec4 &v)
- ostream & [operator<<](#) (ostream &os, vector< float > &v)



### 4.4.1 Detailed Description

Header defining some output methods to print structures to the terminal.

#### Author

Camilo Talero

Version: 0.0.3

### 4.4.2 Function Documentation

#### 4.4.2.1 `operator<<()` [1/4]

```
ostream& operator<< (
    ostream & os,
    vec2 & v )
```

Print a vec2

#### 4.4.2.2 `operator<<()` [2/4]

```
ostream& operator<< (
    ostream & os,
    vec3 & v )
```

Print a vec3

#### 4.4.2.3 `operator<<()` [3/4]

```
ostream& operator<< (
    ostream & os,
    vec4 & v )
```

Print a vec4

#### 4.4.2.4 `operator<<()` [4/4]

```
ostream& operator<< (
    ostream & os,
    vector< float > & v )
```

Print a vector of floats

## 4.5 Helpers/system-libraries.hpp File Reference

General header for system libraries.

```
#include <GL/glew.h>
#include <GLFW/glfw3.h>
#include <string>
#include <sstream>
#include <iostream>
#include <vector>
#include <fstream>
#include <cstdlib>
#include <unistd.h>
#include <time.h>
#include <thread>
#include <mutex>
#include <math.h>
#include <chrono>
#include <ctime>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtx/transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <ft2build.h>
```

### Macros

- `#define GLEW_DYNAMIC`

#### 4.5.1 Detailed Description

General header for system libraries.

#### Author

Camilo Talero

Version: 0.0.3

#### 4.5.2 Macro Definition Documentation

##### 4.5.2.1 GLEW\_DYNAMIC

```
#define GLEW_DYNAMIC
```

## 4.6 Helpers/tools.cpp File Reference

Implementation of miscellaneous helping functions and structures.

```
#include "tools.hpp"
```

### Functions

- void `vec_field_init` ()
- double `fade` (double d)
- double `length` (double x, double y)
- double `surflet` (double x, double y, double grad\_x, double grad\_y)
- double `perlin_noise` (double x, double y)
- double `noise_2D` (double x, double y)

### Variables

- int const `size` = 256
- int const `mask` = `size`-1
- int `perm` [`size`]
- float `vec_field_x` [`size`]
- float `vec_field_y` [`size`]

#### 4.6.1 Detailed Description

Implementation of miscellaneous helping functions and structures.

##### Author

Camilo Talero

Version: 0.0.3

Perlin noise implementation was done following the information at: <http://eastfarthing.com/blog/2015-04-21-noise>

#### 4.6.2 Function Documentation

##### 4.6.2.1 `fade()`

```
double fade (
    double d ) [inline]
```

Function to smooth out the transition from each grid cell to another  $f(x)=1-6*|x|^5-15|x|^4+10|x|^3$

#### 4.6.2.2 length()

```
double length (
    double x,
    double y ) [inline]
```

Return the length of the vector (x,y) for radial fading.

#### 4.6.2.3 noise\_2D()

```
double noise_2D (
    double x,
    double y )
```

Composite 2D noise function. Combines multiple iterations of Perlin noise at different sampling rates and amplitudes and merges them using octaves to create more complex noise functions

#### 4.6.2.4 perlin\_noise()

```
double perlin_noise (
    double x,
    double y )
```

2D Perlin Noise function

#### 4.6.2.5 surflet()

```
double surflet (
    double x,
    double y,
    double grad_x,
    double grad_y ) [inline]
```

2D convolution surflet function, returns a scalar based on the gradient at (x,y)

#### 4.6.2.6 vec\_field\_init()

```
void vec_field_init ( )
```

Initialize the perlin noise grid. We basically rotate a 2D vector 2PI units in the counter clockwise direction and assign a random location to it in a lookup table

### 4.6.3 Variable Documentation

#### 4.6.3.1 mask

```
int const mask = size-1
```

#### 4.6.3.2 perm

```
int perm[size]
```

#### 4.6.3.3 size

```
int const size = 256
```

#### 4.6.3.4 vec\_field\_x

```
float vec_field_x[size]
```

#### 4.6.3.5 vec\_field\_y

```
float vec_field_y[size]
```

## 4.7 Helpers/tools.hpp File Reference

Header for the definition of a generic chunk object.

```
#include "system-libraries.hpp"
```

### Classes

- class [cirArray< T >](#)

### Functions

- double [noise\\_2D](#) (double x, double y)
- void [vec\\_field\\_init](#) ()

### 4.7.1 Detailed Description

Header for the definition of a generic chunk object.

Author

Camilo Talero

Version: 0.0.3

### 4.7.2 Function Documentation

#### 4.7.2.1 noise\_2D()

```
double noise_2D (
    double x,
    double y )
```

Composite 2D noise function. Combines multiple iterations of Perlin noise at different sampling rates and amplitudes and merges them using octaves to create more complex noise functions

#### 4.7.2.2 vec\_field\_init()

```
void vec_field_init ( )
```

Initialize the perlin noise grid. We basically rotate a 2D vector 2PI units in the counter clockwise direction and assign a random location to it in a lookup table

## 4.8 Helpers/wavefront-loader.cpp File Reference

Defines methods needed to load wavefront (.obj) meshes.

```
#include "wavefront-loader.hpp"
#include <algorithm>
```

### Functions

- void [load\\_obj](#) (string filename, vector< float > \*vertices, vector< float > \*normals, vector< float > \*texture\_coords)

### 4.8.1 Detailed Description

Defines methods needed to load wavefront (.obj) meshes.

Author

Camilo Talero

Version: 0.0.3

### 4.8.2 Function Documentation

#### 4.8.2.1 load\_obj()

```
void load_obj (
    string filename,
    vector< float > * vertices,
    vector< float > * normals,
    vector< float > * texture_coords )
```

Function to load the mesh information from a .obj file, it assumes triangular meshes only. All return arrays must be cleared before using the function, else information will be returned at the end of the arrays.

Params: filename: the path to the file to be loaded. vertices: a pointer to a vector of floats where the vertex information will be loaded normals: a pointer to a vector of floats where the normal information will be loaded texture\_coords: a pointer to a vector of floats where the texture mapping information will be loaded

## 4.9 Helpers/wavefront-loader.hpp File Reference

Header declaration of methods needed to load wavefront (.obj) meshes.

```
#include "system-libraries.hpp"
```

### Functions

- void [load\\_obj](#) (std::string filename, std::vector< float > \*vertices, std::vector< float > \*normals, std::vector< float > \*texture\_coords)

#### 4.9.1 Detailed Description

Header declaration of methods needed to load wavefront (.obj) meshes.

Author

: Camilo Talero

Version: 0.0.3

## 4.9.2 Function Documentation

### 4.9.2.1 load\_obj()

```
void load_obj (
    std::string filename,
    std::vector< float > * vertices,
    std::vector< float > * normals,
    std::vector< float > * texture_coords )
```

## 4.10 main.cpp File Reference

main file. Thread and global loop definitoins go here, as well as initialization

```
#include "system-libraries.hpp"
#include "Window-Management.hpp"
#include "Cube.hpp"
#include "World.hpp"
```

### Typedefs

- typedef std::chrono::duration< int, std::ratio< 1, 60 > > [frame\\_duration](#)
- typedef std::chrono::duration< int, std::ratio< 1, 600 > > [world\\_duration](#)

### Functions

- void [render\\_loop](#) (GLFWwindow \*window)
- void [update\\_loop](#) (GLFWwindow \*, GLFWwindow \*)
- int [main](#) (int argc, char \*\*argv)

### 4.10.1 Detailed Description

main file. Thread and global loop definitoins go here, as well as initialization

#### Author

Camilo Talero

Version: 0.0.3

References: <https://open.gl> <http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-ma>  
<http://www.glfw.org/docs/latest/> <http://eastfarthing.com/blog/2015-04-21-noise/>



## 4.10.2 Typedef Documentation

### 4.10.2.1 frame\_duration

```
typedef std::chrono::duration<int, std::ratio<1, 60> > frame_duration
```

### 4.10.2.2 world\_duration

```
typedef std::chrono::duration<int, std::ratio<1, 600> > world_duration
```

## 4.10.3 Function Documentation

### 4.10.3.1 main()

```
int main (
    int argc,
    char ** argv )
```

### 4.10.3.2 render\_loop()

```
void render_loop (
    GLFWwindow * window )
```

### 4.10.3.3 update\_loop()

```
void update_loop (
    GLFWwindow * window,
    GLFWwindow * o_window )
```

## 4.11 Rendering/Camera/Camera.cpp File Reference

Implementation of the camera header. Defines the behaviour for a generic camera.

```
#include "Camera.hpp"
```

### 4.11.1 Detailed Description

Implementation of the camera header. Defines the behaviour for a generic camera.

#### Author

Camilo Talero

Version: 0.0.3

## 4.12 Rendering/Camera/Camera.hpp File Reference

Header declaration of functions and members for a generic camera class.

```
#include "system-libraries.hpp"
```

### Classes

- class [Camera](#)

### 4.12.1 Detailed Description

Header declaration of functions and members for a generic camera class.

#### Author

Camilo Talero

Version: 0.0.3

## 4.13 Rendering/OpenGL-Wrappers.cpp File Reference

Wrapper structures to abstract OpenGL function calls.

```
#include <stb/stb_image.h>
#include <stb/stb_image_write.h>
#include "system-libraries.hpp"
#include "OpenGL-Wrappers.hpp"
```

### Macros

- #define [STB\\_IMAGE\\_IMPLEMENTATION](#)
- #define [STB\\_IMAGE\\_WRITE\\_IMPLEMENTATION](#)

## Functions

- `template<class T >`  
void `init_buffer` (vector< T > data, GLuint buffer, GLenum buffer\_type, GLuint layout, GLboolean normalize, GLenum elements, GLenum data\_type)
- void `verify_uniform_location` (GLint location, string error\_message)

## Variables

- `Renderer * Rendering_Handler`  
*The global render handler.*

### 4.13.1 Detailed Description

Wrapper structures to abstract OpenGL function calls.

#### Author

Camilo Talero

Version: 0.0.3

### 4.13.2 Macro Definition Documentation

#### 4.13.2.1 STB\_IMAGE\_IMPLEMENTATION

```
#define STB_IMAGE_IMPLEMENTATION
```

#### 4.13.2.2 STB\_IMAGE\_WRITE\_IMPLEMENTATION

```
#define STB_IMAGE_WRITE_IMPLEMENTATION
```

### 4.13.3 Function Documentation

#### 4.13.3.1 init\_buffer()

```
template<class T >
void init_buffer (
    vector< T > data,
    GLuint buffer,
    GLenum buffer_type,
    GLuint layout,
    GLboolean normalize,
    GLuint elements,
    GLenum data_type ) [inline]
```

Method to initialize a basic [Shader](#) layout from a vector of data. Mainly use to make the code less verbose.

#### 4.13.3.2 verify\_uniform\_location()

```
void verify_uniform_location (
    GLint location,
    string error_message ) [inline]
```

Error checking and message function for uniforms.

### 4.13.4 Variable Documentation

#### 4.13.4.1 Rendering\_Handler

[Renderer\\*](#) Rendering\_Handler

The global render handler.

## 4.14 Rendering/OpenGL-Wrappers.hpp File Reference

Header to define variables, structure definitions, include libraries... Shared among all rendering functions.

```
#include "system-libraries.hpp"
#include "Camera.hpp"
#include "cout-definitions.hpp"
```

### Classes

- class [Shader](#)
- class [Texture](#)
- struct [Mesh](#)
- class [Renderer](#)
- class [Object\\_3D](#)

## Enumerations

- enum [PROGRAM](#)

## Functions

- int [openGLError](#) ()

## Variables

- [Renderer](#) \* [Rendering\\_Handler](#)

*The global render handler.*

### 4.14.1 Detailed Description

Header to define variables, structure definitions, include libraries... Shared among all rendering functions.

#### Author

Camilo Talero

Version: 0.0.3

### 4.14.2 Enumeration Type Documentation

#### 4.14.2.1 PROGRAM

enum [PROGRAM](#)

### 4.14.3 Function Documentation

#### 4.14.3.1 [openGLError](#)()

```
int openGLError ( )
```

Check for OpenGL errors and print the appropriate error message if needed.

#### Returns

The number of the generated error.

#### 4.14.4 Variable Documentation

##### 4.14.4.1 Rendering\_Handler

`Renderer*` `Rendering_Handler`

The global render handler.

### 4.15 Rendering/Window-Management.cpp File Reference

File defining all relevant OpenGL and GLFW related functions needed to create an OpenGL context and GLFW window.

```
#include "Window-Management.hpp"
```

#### Macros

- `#define CAM_SPEED 0.3f`

#### Functions

- `GLFWwindow *` `create_context` (`GLFWwindow *``other_window`, `bool visible`)
- `int` `openGLError` ()
- `void` `callBackInit` (`GLFWwindow *``window`)
- `GLFWwindow *` `createWindow` (`GLFWwindow *``other_window`, `bool visible`)
- `void` `error_callback` (`int error`, `const char *``description`)
- `void` `cursor_pos_callback` (`GLFWwindow *``window`, `double xpos`, `double ypos`)
- `void` `mouse_button_callback` (`GLFWwindow *``window`, `int button`, `int action`, `int mods`)
- `void` `key_callback` (`GLFWwindow *``window`, `int key`, `int scancode`, `int action`, `int mods`)

#### 4.15.1 Detailed Description

File defining all relevant OpenGL and GLFW related functions needed to create an OpenGL context and GLFW window.

#### Author

Camilo Talero

Version: 0.0.3

#### 4.15.2 Macro Definition Documentation

#### 4.15.2.1 CAM\_SPEED

```
#define CAM_SPEED 0.3f
```

### 4.15.3 Function Documentation

#### 4.15.3.1 callBackInit()

```
void callBackInit (
    GLFWwindow * window )
```

Initialize GLFW callBack Functions

#### 4.15.3.2 create\_context()

```
GLFWwindow* create_context (
    GLFWwindow * other_window,
    bool visible )
```

Function to create the OpenGL context.

##### Returns

The pointer to the GLFW window containing the current context.

#### 4.15.3.3 createWindow()

```
GLFWwindow* createWindow (
    GLFWwindow * other_window,
    bool visible )
```

Method to create a GLFW window, window will be maximized and decorated.

##### Returns

A pointer to the created window.

#### 4.15.3.4 cursor\_pos\_callback()

```
void cursor_pos_callback (
    GLFWwindow * window,
    double xpos,
    double ypos )
```

GLFW cursor position function

#### 4.15.3.5 error\_callback()

```
void error_callback (
    int error,
    const char * description )
```

Print out GLFW error information

#### 4.15.3.6 key\_callback()

```
void key_callback (
    GLFWwindow * window,
    int key,
    int scancode,
    int action,
    int mods )
```

GLFW keys function

Called when a key is pressed and handles the event for each implemented key

#### 4.15.3.7 mouse\_button\_callback()

```
void mouse_button_callback (
    GLFWwindow * window,
    int button,
    int action,
    int mods )
```

GLFW Mouse button function

#### 4.15.3.8 openGLError()

```
int openGLError ( )
```

Check for OpenGL errors and print the appropriate error message if needed.

##### Returns

The number of the generated error.

## 4.16 Rendering/Window-Management.hpp File Reference

Header for the context creation implementation. Exposes functions and defines needed included files.

```
#include "system-libraries.hpp"
#include "OpenGL-Wrappers.hpp"
```



## Functions

- void [error\\_callback](#) (int error, const char \*description)
- void [key\\_callback](#) (GLFWwindow \*window, int key, int scancode, int action, int mods)
- void [mouse\\_button\\_callback](#) (GLFWwindow \*window, int button, int action, int mods)
- void [cursor\\_pos\\_callback](#) (GLFWwindow \*window, double xpos, double ypos)
- void [callBackInit](#) (GLFWwindow \*window)
- double [calculateFPS](#) (double prevTime, double currentTime)
- GLFWwindow \* [createWindow](#) (GLFWwindow \*other\_window, bool)
- GLFWwindow \* [create\\_context](#) (GLFWwindow \*other\_window, bool)

### 4.16.1 Detailed Description

Header for the context creation implementation. Exposes functions and defines needed included files.

#### Author

Camilo Talero

Version: 0.0.3

### 4.16.2 Function Documentation

#### 4.16.2.1 [calculateFPS\(\)](#)

```
double calculateFPS (
    double prevTime,
    double currentTime )
```

#### 4.16.2.2 [callBackInit\(\)](#)

```
void callBackInit (
    GLFWwindow * window )
```

Initialize GLFW callBack Functions

#### 4.16.2.3 [create\\_context\(\)](#)

```
GLFWwindow* create_context (
    GLFWwindow * other_window,
    bool visible )
```

Function to create the OpenGL context.

#### Returns

The pointer to the GLFW window containing the current context.

#### 4.16.2.4 `createWindow()`

```
GLFWwindow* createWindow (
    GLFWwindow * other_window,
    bool visible )
```

Method to create a GLFW window, window will be maximized and decorated.

##### Returns

A pointer to the created window.

#### 4.16.2.5 `cursor_pos_callback()`

```
void cursor_pos_callback (
    GLFWwindow * window,
    double xpos,
    double ypos )
```

GLFW cursor position function

#### 4.16.2.6 `error_callback()`

```
void error_callback (
    int error,
    const char * description )
```

Print out GLFW error information

#### 4.16.2.7 `key_callback()`

```
void key_callback (
    GLFWwindow * window,
    int key,
    int scancode,
    int action,
    int mods )
```

GLFW keys function

Called when a key is pressed and handles the event for each implemented key

#### 4.16.2.8 `mouse_button_callback()`

```
void mouse_button_callback (
    GLFWwindow * window,
    int button,
    int action,
    int mods )
```

GLFW Mouse button function

## 4.17 World.cpp File Reference

Definitions of all world related classes and methods.

```
#include "World.hpp"
#include "cout-definitions.hpp"
```

### Macros

- #define `MESH Cube::meshes[0]`

### Variables

- `World * the_world`  
*Global world object, should eb treated as a singleton.*

### 4.17.1 Detailed Description

Definitions of all world related classes and methods.

#### Author

Camilo Talero

Version: 0.0.3

### 4.17.2 Macro Definition Documentation

#### 4.17.2.1 MESH

```
#define MESH Cube::meshes[0]
```

### 4.17.3 Variable Documentation

#### 4.17.3.1 the\_world

```
World* the_world
```

Global world object, should eb treated as a singleton.

## 4.18 World.hpp File Reference

Header for the definition of a generic chunk object.

```
#include "Cube.hpp"  
#include "tools.hpp"
```

### Classes

- struct [Light](#)
- class [Chunk](#)
- class [Chunk\\_Holder](#)
- class [World](#)

### Macros

- `#define` [CHUNK\\_DIMS](#) 16

### Variables

- [World](#) \* [the\\_world](#)  
*Global world object, should eb treated as a singleton.*

### 4.18.1 Detailed Description

Header for the definition of a generic chunk object.

#### Author

Camilo Talero

Version: 0.0.3

### 4.18.2 Macro Definition Documentation

#### 4.18.2.1 CHUNK\_DIMS

```
#define CHUNK_DIMS 16
```

### 4.18.3 Variable Documentation

#### 4.18.3.1 the\_world

```
World* the\_world
```

Global world object, should eb treated as a singleton.

# Index

- ~Camera
  - Camera, [6](#)
- ~Chunk
  - Chunk, [12](#)
- ~Chunk\_Holder
  - Chunk\_Holder, [15](#)
- ~Cube
  - Cube, [19](#)
- ~Mesh
  - Mesh, [22](#)
- ~Renderer
  - Renderer, [27](#)
- ~Shader
  - Shader, [31](#)
- ~Texture
  - Texture, [33](#)
- ~World
  - World, [36](#)
- add\_Shader
  - Renderer, [27](#)
- add\_data
  - Renderer, [27](#)
- array
  - cirArray, [18](#)
- busy\_queue
  - Renderer, [29](#)
- CAM\_SPEED
  - Window-Management.cpp, [56](#)
- CHUNK\_DIMS
  - World.hpp, [62](#)
- calculateFPS
  - Window-Management.hpp, [59](#)
- callBackInit
  - Window-Management.cpp, [57](#)
  - Window-Management.hpp, [59](#)
- cam
  - Renderer, [29](#)
- Camera, [5](#)
  - ~Camera, [6](#)
  - Camera, [6](#)
  - forward, [9](#)
  - fov, [9](#)
  - getForward, [6](#)
  - getFov, [7](#)
  - getPerspectiveMatrix, [7](#)
  - getPosition, [7](#)
  - getSide, [7](#)
  - getUp, [7](#)
  - getViewMatrix, [7](#)
  - height, [9](#)
  - incline, [7](#)
  - move, [7](#)
  - orig\_forward, [9](#)
  - orig\_position, [9](#)
  - orig\_side, [9](#)
  - orig\_up, [9](#)
  - position, [10](#)
  - resetCamera, [8](#)
  - resetView, [8](#)
  - setLookDirection, [8](#)
  - setPosition, [8](#)
  - side, [10](#)
  - turnH, [8](#)
  - turnV, [8](#)
  - up, [10](#)
  - width, [10](#)
  - zFar, [10](#)
  - zNear, [10](#)
- center\_frame
  - World, [36](#)
- change\_active\_program
  - Renderer, [27](#)
- check\_neighbour
  - Chunk, [12](#)
- Chunk, [11](#)
  - ~Chunk, [12](#)
  - check\_neighbour, [12](#)
  - Chunk, [11](#), [12](#)
  - chunk\_cubes, [13](#)
  - create\_cubes, [12](#)
  - cubes\_info, [13](#)
  - operator(), [12](#)
  - position, [13](#)
  - render\_data, [14](#)
  - send\_render\_data, [13](#)
  - update, [13](#)
  - update\_visible\_cubes, [13](#)
  - world, [14](#)
- Chunk\_Holder, [14](#)
  - ~Chunk\_Holder, [15](#)
  - Chunk\_Holder, [14](#), [15](#)
  - chunkBox, [15](#)
  - operator(), [15](#)
  - shift, [15](#)
  - world, [16](#)
- chunk\_cubes

- Chunk, 13
- chunkBox
  - Chunk\_Holder, 15
- cirArray
  - array, 18
  - cirArray, 16, 17
  - operator=, 17
  - operator[], 17
  - shift, 17
  - size, 17
  - start, 18
- cirArray< T >, 16
- cleanup
  - Cube, 19
- clear
  - Renderer, 27
  - Shader, 31
  - Texture, 34
- color
  - Light, 21
- cout-definitions.cpp
  - operator<<, 41, 42
- cout-definitions.hpp
  - operator<<, 43
- create\_context
  - Window-Management.cpp, 57
  - Window-Management.hpp, 59
- create\_cubes
  - Chunk, 12
- createWindow
  - Window-Management.cpp, 57
  - Window-Management.hpp, 59
- Cube, 18
  - ~Cube, 19
  - cleanup, 19
  - Cube, 19
  - cube\_type, 20
  - getMesh, 20
  - initialize, 20
  - meshes, 20
  - position, 20
  - textures, 21
  - transparent, 21
  - update, 20
- Cube.cpp, 39
  - obj\_source\_files, 39
  - texture\_source\_files, 39
- Cube.hpp, 40
  - cube\_types, 41
  - CubeID, 40
- cube\_type
  - Cube, 20
- cube\_types
  - Cube.hpp, 41
- CubeID
  - Cube.hpp, 40
- cubes\_info
  - Chunk, 13
- current\_program
  - Renderer, 29
- cursor\_pos\_callback
  - Window-Management.cpp, 57
  - Window-Management.hpp, 60
- error\_callback
  - Window-Management.cpp, 57
  - Window-Management.hpp, 60
- fade
  - tools.cpp, 45
- fileName
  - Shader, 32
- find\_shader
  - Renderer, 28
- forward
  - Camera, 9
- fov
  - Camera, 9
- fragment\_shaders
  - Renderer, 29
- frame\_duration
  - main.cpp, 51
- GLEW\_DYNAMIC
  - system-libraries.hpp, 44
- getForward
  - Camera, 6
- getFov
  - Camera, 7
- getMesh
  - Cube, 20
- getPerspectiveMatrix
  - Camera, 7
- getPosition
  - Camera, 7
- getSide
  - Camera, 7
- getUp
  - Camera, 7
- getViewMatrix
  - Camera, 7
- h\_radius
  - World, 36
- height
  - Camera, 9
  - Texture, 34
- Helpers/cout-definitions.cpp, 41
- Helpers/cout-definitions.hpp, 42
- Helpers/system-libraries.hpp, 44
- Helpers/tools.cpp, 45
- Helpers/tools.hpp, 47
- Helpers/wavefront-loader.cpp, 48
- Helpers/wavefront-loader.hpp, 49
- incline
  - Camera, 7

- indices
  - Mesh, [22](#)
- init\_buffer
  - OpenGL-Wrappers.cpp, [53](#)
- initialize
  - Cube, [20](#)
- intensity
  - Light, [21](#)
- key\_callback
  - Window-Management.cpp, [58](#)
  - Window-Management.hpp, [60](#)
- layouts
  - Object\_3D, [24](#)
- length
  - tools.cpp, [45](#)
- Light, [21](#)
  - color, [21](#)
  - intensity, [21](#)
  - position, [22](#)
- load\_from\_file
  - Shader, [31](#)
- load\_obj
  - wavefront-loader.cpp, [49](#)
  - wavefront-loader.hpp, [50](#)
- load\_to\_GPU
  - Texture, [34](#)
- loaded\_chunks
  - World, [37](#)
- MESH
  - World.cpp, [61](#)
- main
  - main.cpp, [51](#)
- main.cpp, [50](#)
  - frame\_duration, [51](#)
  - main, [51](#)
  - render\_loop, [51](#)
  - update\_loop, [51](#)
  - world\_duration, [51](#)
- make\_program
  - Renderer, [28](#)
- mask
  - tools.cpp, [46](#)
- Mesh, [22](#)
  - ~Mesh, [22](#)
  - indices, [22](#)
  - normals, [23](#)
  - uvs, [23](#)
  - vertices, [23](#)
- mesh\_indices
  - Object\_3D, [25](#)
- meshes
  - Cube, [20](#)
- mouse\_button\_callback
  - Window-Management.cpp, [58](#)
  - Window-Management.hpp, [60](#)
- move
  - Camera, [7](#)
- multi\_render
  - Renderer, [28](#)
- noise\_2D
  - tools.cpp, [46](#)
  - tools.hpp, [48](#)
- normals
  - Mesh, [23](#)
- obj\_source\_files
  - Cube.cpp, [39](#)
- Object\_3D, [23](#)
  - layouts, [24](#)
  - mesh\_indices, [25](#)
  - Object\_3D, [24](#)
  - render\_instances, [25](#)
  - set\_instance\_data, [24](#)
  - types, [25](#)
  - VAO, [25](#)
  - VBOs, [25](#)
- OpenGL-Wrappers.cpp
  - init\_buffer, [53](#)
  - Rendering\_Handler, [54](#)
  - STB\_IMAGE\_IMPLEMENTATION, [53](#)
  - STB\_IMAGE\_WRITE\_IMPLEMENTATION, [53](#)
  - verify\_uniform\_location, [54](#)
- OpenGL-Wrappers.hpp
  - openGLError, [55](#)
  - PROGRAM, [55](#)
  - Rendering\_Handler, [56](#)
- openGLError
  - OpenGL-Wrappers.hpp, [55](#)
  - Window-Management.cpp, [58](#)
- operator<<
  - cout-definitions.cpp, [41](#), [42](#)
  - cout-definitions.hpp, [43](#)
- operator()
  - Chunk, [12](#)
  - Chunk\_Holder, [15](#)
  - World, [36](#)
- operator=
  - cirArray, [17](#)
- operator[]
  - cirArray, [17](#)
- orig\_forward
  - Camera, [9](#)
- orig\_position
  - Camera, [9](#)
- orig\_side
  - Camera, [9](#)
- orig\_up
  - Camera, [9](#)
- origin
  - World, [37](#)
- PROGRAM
  - OpenGL-Wrappers.hpp, [55](#)
- perlin\_noise

- tools.cpp, 46
- perm
  - tools.cpp, 47
- position
  - Camera, 10
  - Chunk, 13
  - Cube, 20
  - Light, 22
- render
  - Renderer, 28
- render\_data
  - Chunk, 14
- render\_instances
  - Object\_3D, 25
- render\_loop
  - main.cpp, 51
- render\_queue
  - Renderer, 29
- Renderer, 26
  - ~Renderer, 27
  - add\_Shader, 27
  - add\_data, 27
  - busy\_queue, 29
  - cam, 29
  - change\_active\_program, 27
  - clear, 27
  - current\_program, 29
  - find\_shader, 28
  - fragment\_shaders, 29
  - make\_program, 28
  - multi\_render, 28
  - render, 28
  - render\_queue, 29
  - Renderer, 26, 27
  - set\_camera, 28
  - shading\_programs, 29
  - tessellation\_shaders, 30
  - update, 28
  - vertex\_shaders, 30
- Rendering/Camera/Camera.cpp, 51
- Rendering/Camera/Camera.hpp, 52
- Rendering/OpenGL-Wrappers.cpp, 52
- Rendering/OpenGL-Wrappers.hpp, 54
- Rendering/Window-Management.cpp, 56
- Rendering/Window-Management.hpp, 58
- Rendering\_Handler
  - OpenGL-Wrappers.cpp, 54
  - OpenGL-Wrappers.hpp, 56
- resetCamera
  - Camera, 8
- resetView
  - Camera, 8
- STB\_IMAGE\_IMPLEMENTATION
  - OpenGL-Wrappers.cpp, 53
- STB\_IMAGE\_WRITE\_IMPLEMENTATION
  - OpenGL-Wrappers.cpp, 53
- send\_render\_data
  - Chunk, 13
  - World, 36
- set\_camera
  - Renderer, 28
- set\_instance\_data
  - Object\_3D, 24
- setLookDirection
  - Camera, 8
- setPosition
  - Camera, 8
- Shader, 30
  - ~Shader, 31
  - clear, 31
  - fileName, 32
  - load\_from\_file, 31
  - Shader, 31
  - shaderID, 32
  - type, 32
- shaderID
  - Shader, 32
- shading\_programs
  - Renderer, 29
- shift
  - Chunk\_Holder, 15
  - cirArray, 17
- side
  - Camera, 10
- size
  - cirArray, 17
  - tools.cpp, 47
- start
  - cirArray, 18
- surflet
  - tools.cpp, 46
- system-libraries.hpp
  - GLEW\_DYNAMIC, 44
- target
  - Texture, 34
- tessellation\_shaders
  - Renderer, 30
- Texture, 32
  - ~Texture, 33
  - clear, 34
  - height, 34
  - load\_to\_GPU, 34
  - target, 34
  - Texture, 33
  - texture, 34
  - textureID, 34
  - width, 35
- texture
  - Texture, 34
- texture\_source\_files
  - Cube.cpp, 39
- textureID
  - Texture, 34
- textures
  - Cube, 21



- the\_world
  - World.cpp, 61
  - World.hpp, 62
- tools.cpp
  - fade, 45
  - length, 45
  - mask, 46
  - noise\_2D, 46
  - perlin\_noise, 46
  - perm, 47
  - size, 47
  - surflet, 46
  - vec\_field\_init, 46
  - vec\_field\_x, 47
  - vec\_field\_y, 47
- tools.hpp
  - noise\_2D, 48
  - vec\_field\_init, 48
- transparent
  - Cube, 21
- turnH
  - Camera, 8
- turnV
  - Camera, 8
- type
  - Shader, 32
- types
  - Object\_3D, 25
- up
  - Camera, 10
- update
  - Chunk, 13
  - Cube, 20
  - Renderer, 28
- update\_loop
  - main.cpp, 51
- update\_visible\_cubes
  - Chunk, 13
- uvs
  - Mesh, 23
- v\_radius
  - World, 37
- VAO
  - Object\_3D, 25
- VBOs
  - Object\_3D, 25
- vec\_field\_init
  - tools.cpp, 46
  - tools.hpp, 48
- vec\_field\_x
  - tools.cpp, 47
- vec\_field\_y
  - tools.cpp, 47
- verify\_uniform\_location
  - OpenGL-Wrappers.cpp, 54
- vertex\_shaders
  - Renderer, 30
- vertices
  - Mesh, 23
- wavefront-loader.cpp
  - load\_obj, 49
- wavefront-loader.hpp
  - load\_obj, 50
- width
  - Camera, 10
  - Texture, 35
- Window-Management.cpp
  - CAM\_SPEED, 56
  - callBackInit, 57
  - create\_context, 57
  - createWindow, 57
  - cursor\_pos\_callback, 57
  - error\_callback, 57
  - key\_callback, 58
  - mouse\_button\_callback, 58
  - openGLError, 58
- Window-Management.hpp
  - calculateFPS, 59
  - callBackInit, 59
  - create\_context, 59
  - createWindow, 59
  - cursor\_pos\_callback, 60
  - error\_callback, 60
  - key\_callback, 60
  - mouse\_button\_callback, 60
- World, 35
  - ~World, 36
  - center\_frame, 36
  - h\_radius, 36
  - loaded\_chunks, 37
  - operator(), 36
  - origin, 37
  - send\_render\_data, 36
  - v\_radius, 37
  - World, 35
- world
  - Chunk, 14
  - Chunk\_Holder, 16
- World.cpp, 61
  - MESH, 61
  - the\_world, 61
- World.hpp, 62
  - CHUNK\_DIMS, 62
  - the\_world, 62
- world\_duration
  - main.cpp, 51
- zFar
  - Camera, 10
- zNear
  - Camera, 10