

PointCloud2BIM Library

1.0

Generated by Doxygen 1.8.11

Contents

1	PointCloud2BIM Library reference manual	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Class Documentation	9
5.1	BitGrid Class Reference	9
5.1.1	Constructor & Destructor Documentation	10
5.1.1.1	BitGrid(int dimensions, struct BoundingBoxInfo &bbox, double step)	10
5.1.2	Member Function Documentation	10
5.1.2.1	add(PointXYZNormal &p)	11
5.1.2.2	add(PointXYZ &p)	11
5.1.2.3	add(int i)	11
5.1.2.4	add(int x, int y)	11
5.1.2.5	get(int i)	12
5.1.2.6	get(int x, int y)	12
5.1.2.7	get(int x, int y, int z)	12
5.1.2.8	getActiveCells()	12
5.1.2.9	getActiveCellsIndex(std::vector< int > &res)	13

5.1.2.10	<code>getArea()</code>	13
5.1.2.11	<code>getBBox()</code>	13
5.1.2.12	<code>getBoundingPolygon()</code>	13
5.1.2.13	<code>getDimensions()</code>	13
5.1.2.14	<code>getPoint(int i)</code>	13
5.1.2.15	<code>getPoints()</code>	14
5.1.2.16	<code>getSize()</code>	14
5.1.2.17	<code>getSizeX()</code>	14
5.1.2.18	<code>getSizeY()</code>	14
5.1.2.19	<code>getSizeZ()</code>	14
5.1.2.20	<code>getStepX()</code>	15
5.1.2.21	<code>getStepY()</code>	15
5.1.2.22	<code>getStepZ()</code>	15
5.1.2.23	<code>getX(int i)</code>	15
5.1.2.24	<code>getY(int i)</code>	15
5.1.2.25	<code>pointExists(PointXYZ &p)</code>	16
5.1.2.26	<code>remove(int i)</code>	16
5.1.2.27	<code>toSVG(std::filesystem::path path)</code>	16
5.2	BoundingBoxInfo Struct Reference	16
5.2.1	Member Function Documentation	17
5.2.1.1	<code>giveBoundingBoxMax(double(&max)[3])</code>	17
5.2.1.2	<code>giveBoundingBoxMin(double(&min)[3])</code>	17
5.2.1.3	<code>range(int index)</code>	17
5.3	Cell Struct Reference	18
5.3.1	Member Function Documentation	18
5.3.1.1	<code>operator==(const Cell &o) const</code>	18
5.3.2	Friends And Related Function Documentation	19
5.3.2.1	<code>hash_value</code>	19
5.4	Downsampler Class Reference	19
5.4.1	Constructor & Destructor Documentation	20

5.4.1.1	Downsampler(double step)	20
5.4.1.2	Downsampler(double x, double y, double z)	21
5.4.2	Member Function Documentation	21
5.4.2.1	addPoint(PointXYZ &p)	21
5.4.2.2	downsample(BoundingBoxInfo &bb, std::vector< PointXYZ > &points)	21
5.4.2.3	getIndex(int x, int y, int z)	21
5.4.2.4	getPoints(std::vector< PointXYZ > &pts)	22
5.4.2.5	getPoints(std::vector< PointXYZNormal > &pts)	22
5.4.2.6	getSize()	22
5.4.2.7	isActive(int i)	22
5.4.2.8	setSizes(BoundingBoxInfo &bb)	22
5.4.2.9	setStep(double step)	23
5.4.2.10	setStep(double x, double y, double z)	23
5.5	Floor Class Reference	23
5.5.1	Member Function Documentation	24
5.5.1.1	addRoom(Room &room)	24
5.5.1.2	addWall(Wall &wall)	24
5.5.1.3	getCeilingLevel() const	25
5.5.1.4	getFloorLevel() const	25
5.5.1.5	getId() const	25
5.5.1.6	getMax() const	25
5.5.1.7	getMin() const	25
5.5.1.8	getPoints(std::vector< T > &points)	25
5.5.1.9	getRoomIDs()	26
5.5.1.10	getRooms() const	26
5.5.1.11	getRooms()	26
5.5.1.12	getWallIDs()	26
5.5.1.13	getWalls() const	26
5.5.1.14	getWalls()	27
5.5.1.15	setCeilingLevel(double l)	27

5.5.1.16	setFloorLevel(double l)	27
5.5.1.17	setId(int id)	27
5.5.1.18	setMax(double x, double y, double z)	27
5.5.1.19	setMin(double x, double y, double z)	28
5.5.1.20	setRoomIDs(std::vector< int > room_ids)	28
5.5.1.21	setWallIDs(std::vector< int > wall_ids)	28
5.6	Histogram Class Reference	28
5.6.1	Constructor & Destructor Documentation	29
5.6.1.1	Histogram(BoundingBoxInfo &bbox, double step, int coord)	29
5.6.2	Member Function Documentation	29
5.6.2.1	addPoint(PointXYZ &p)	29
5.6.2.2	addPoint(PointXYZNormal &p)	29
5.6.2.3	computeHistogram(std::vector< PointXYZ > &points, BoundingBoxInfo &bbox, double step, int coord)	29
5.6.2.4	getHistogram()	30
5.7	HistogramInfo Struct Reference	30
5.8	Opening Class Reference	31
5.8.1	Member Function Documentation	31
5.8.1.1	addPointCloud(PointCloud &pc)	31
5.8.1.2	getCornersX() const	32
5.8.1.3	getCornersY() const	32
5.8.1.4	getCornersZ() const	32
5.8.1.5	getDimensions() const	32
5.8.1.6	getDimensions()	32
5.8.1.7	getId() const	33
5.8.1.8	getOpeningType() const	33
5.8.1.9	getPointCloudIDs()	33
5.8.1.10	getPointClouds() const	33
5.8.1.11	getPointClouds()	33
5.8.1.12	getPoints(std::vector< T > &points)	33
5.8.1.13	setCornersX(std::vector< double > cornersX)	34

5.8.1.14	setCornersY(std::vector< double > cornersY)	34
5.8.1.15	setCornersZ(std::vector< double > cornersZ)	34
5.8.1.16	setDimensions(std::vector< double > dimensions)	34
5.8.1.17	setId(int id)	35
5.8.1.18	setOpeningType(OpeningType name)	35
5.8.1.19	setPointCloudIDs(std::vector< int > pcl_ids)	35
5.9	Plane Class Reference	35
5.9.1	Constructor & Destructor Documentation	37
5.9.1.1	Plane(double _a, double _b, double _c, double _d)	37
5.9.1.2	Plane(Vector3 _a, Vector3 _b, Vector3 _c)	38
5.9.1.3	Plane(PointXYZ _a, PointXYZ _b, PointXYZ _c)	38
5.9.1.4	Plane(PointXYZNormal _a, PointXYZNormal _b, PointXYZNormal _c)	38
5.9.2	Member Function Documentation	38
5.9.2.1	addPointCloud(PointCloud &pc)	39
5.9.2.2	checkLocalMatch(std::vector< PointXYZNormal > points1, std::vector< PointXYZNormal > points2)	39
5.9.2.3	checkMatchingArea(std::vector< PointXYZNormal > points1, std::vector< PointXYZNormal > points2)	39
5.9.2.4	distanceBetweenPlanes(Plane &pl1, Plane &pl2)	39
5.9.2.5	distanceTo(Vector3 &v)	40
5.9.2.6	distanceTo(PointXYZ &p)	40
5.9.2.7	distanceTo(PointXYZNormal p)	40
5.9.2.8	distanceTo(std::vector< PointXYZNormal > &points)	40
5.9.2.9	generateOuterPlane(std::vector< PointXYZNormal > points, std::vector< PointXYZNormal > &outPoints, std::vector< BitGrid > roomGrids, double minOffset)	41
5.9.2.10	getD() const	41
5.9.2.11	getFloorId() const	41
5.9.2.12	getId() const	41
5.9.2.13	getLocalMatch(std::vector< PointXYZNormal > points1, std::vector< PointXYZNormal > points2, std::vector< int > &result1, std::vector< int > &result2)	42
5.9.2.14	getMax() const	42
5.9.2.15	getMin() const	42

5.9.2.16	<code>getNormal() const</code>	42
5.9.2.17	<code>getNormal()</code>	42
5.9.2.18	<code>getNormalVector()</code>	43
5.9.2.19	<code>getPointCloudIDs()</code>	43
5.9.2.20	<code>getPointClouds() const</code>	43
5.9.2.21	<code>getPointClouds()</code>	43
5.9.2.22	<code>getPoints(std::vector< T > &points)</code>	43
5.9.2.23	<code>giveCenterPlane(std::vector< PointXYZNormal > points1, std::vector< PointXYZNormal > points2, double(&v1)[3], double(&v2)[3], double(&v3)[3], double &d, std::vector< PointXYZNormal > &corners)</code>	43
5.9.2.24	<code>isInnerPlane(std::vector< PointXYZNormal > points, std::vector< BitGrid > roomGrids, double minOffset)</code>	44
5.9.2.25	<code>makeWall(Plane &plane1, Plane &plane2, std::vector< PointXYZ > &result1, std::vector< PointXYZ > &result2)</code>	44
5.9.2.26	<code>setD(double d)</code>	44
5.9.2.27	<code>setFloorId(int id)</code>	45
5.9.2.28	<code>setId(int id)</code>	45
5.9.2.29	<code>setMax(double x, double y, double z)</code>	45
5.9.2.30	<code>setMin(double x, double y, double z)</code>	45
5.9.2.31	<code>setNormal(double nx, double ny, double nz)</code>	45
5.10	<code>Point< Dimensions > Class Template Reference</code>	46
5.10.1	Member Function Documentation	46
5.10.1.1	<code>getCoords()</code>	46
5.10.1.2	<code>getId()</code>	46
5.10.1.3	<code>setId(double id)</code>	46
5.11	<code>PointCloud Class Reference</code>	47
5.11.1	Constructor & Destructor Documentation	48
5.11.1.1	<code>PointCloud(std::string name)</code>	48
5.11.2	Member Function Documentation	48
5.11.2.1	<code>getFilepath()</code>	48
5.11.2.2	<code>getId() const</code>	48
5.11.2.3	<code>getName() const</code>	48

5.11.2.4	getNumberOfPoints()	48
5.11.2.5	getPoints(std::vector< T > &points)	48
5.11.2.6	getPointType() const	49
5.11.2.7	readXYZNormalPoint(std::ifstream &infile, std::vector< PointXYZNormal > &points)	49
5.11.2.8	readXYZNormalPoint(std::ifstream &infile, std::vector< PointXYZ > &points)	49
5.11.2.9	readXYZPoint(std::ifstream &infile, std::vector< PointXYZ > &points)	49
5.11.2.10	readXYZPoint(std::ifstream &infile, std::vector< PointXYZNormal > &points)	50
5.11.2.11	setFilepath(fs::path filepath)	50
5.11.2.12	setId(double id)	50
5.11.2.13	setName(std::string name)	50
5.11.2.14	setPoints(std::vector< T > &points)	50
5.11.2.15	setPointType(PointType name)	51
5.11.2.16	writePointsCount(int n)	51
5.11.2.17	writeXYZNormalPoint(std::fstream &output, PointXYZ &point)	51
5.11.2.18	writeXYZNormalPoint(std::fstream &output, PointXYZNormal &point)	51
5.11.2.19	writeXYZPoint(std::fstream &output, T &point)	51
5.12	PointNormal Class Reference	52
5.12.1	Constructor & Destructor Documentation	53
5.12.1.1	PointNormal(const PointNormal &p2)	53
5.12.1.2	PointNormal(double x, double y, double z)	54
5.12.2	Member Function Documentation	54
5.12.2.1	getNormal()	54
5.12.2.2	getNx()	54
5.12.2.3	getNy()	54
5.12.2.4	getNz()	54
5.12.2.5	setNx(double x)	54
5.12.2.6	setNy(double y)	55
5.12.2.7	setNz(double z)	55
5.13	PointXY Class Reference	55
5.13.1	Constructor & Destructor Documentation	56

5.13.1.1	PointXY(double x, double y)	56
5.13.2	Member Function Documentation	56
5.13.2.1	getX()	56
5.13.2.2	getY()	57
5.13.2.3	setX(double x)	57
5.13.2.4	setY(double y)	57
5.14	PointXYZ Class Reference	58
5.14.1	Constructor & Destructor Documentation	59
5.14.1.1	PointXYZ(const PointXYZ &p2)	59
5.14.1.2	PointXYZ(double x, double y, double z)	59
5.14.2	Member Function Documentation	59
5.14.2.1	getX()	59
5.14.2.2	getY()	59
5.14.2.3	getZ()	60
5.14.2.4	setCoords(double x, double y, double z)	60
5.14.2.5	setX(double x)	60
5.14.2.6	setY(double y)	60
5.14.2.7	setZ(double z)	60
5.15	PointXYZInfo Struct Reference	61
5.16	PointXYZNormal Class Reference	61
5.16.1	Constructor & Destructor Documentation	62
5.16.1.1	PointXYZNormal(double x, double y, double z, double nx, double ny, double nz)	62
5.16.1.2	PointXYZNormal(const PointXYZNormal &p2)	63
5.17	PointXYZNormalInfo Struct Reference	63
5.18	Polygon Class Reference	63
5.18.1	Member Function Documentation	63
5.18.1.1	addPoint(PointXY &point)	63
5.18.1.2	isPointInside(PointXYZ &point)	64
5.18.1.3	isPointInside(PointXY &point)	64
5.19	Project Class Reference	64

5.19.1	Constructor & Destructor Documentation	66
5.19.1.1	Project(const fs::path &directory)	66
5.19.1.2	Project(std::string project_name, const fs::path &full)	66
5.19.2	Member Function Documentation	66
5.19.2.1	addFloor(Floor &f, bool force_save=true)	66
5.19.2.2	addOpening(Opening &p, bool force_save=true)	67
5.19.2.3	addPlane(Plane &p, bool force_save=true)	67
5.19.2.4	addPointCloud(PointCloud &pc, bool force_save=true)	67
5.19.2.5	addRoom(Room &r, bool force_save=true)	67
5.19.2.6	addWall(Wall &p, bool force_save=true)	67
5.19.2.7	createProjectFile(const fs::path &path)	68
5.19.2.8	getFloor(int id)	68
5.19.2.9	getFloors()	68
5.19.2.10	getFolder()	68
5.19.2.11	getOpening(int id)	68
5.19.2.12	getOpenings()	69
5.19.2.13	getPlane(int id)	69
5.19.2.14	getPlanes()	69
5.19.2.15	getPointCloud(int id)	69
5.19.2.16	getPointClouds()	70
5.19.2.17	getRoom(int id)	70
5.19.2.18	getRooms()	70
5.19.2.19	getTranslation()	70
5.19.2.20	getWall(int id)	70
5.19.2.21	getWalls()	71
5.19.2.22	import(std::string filename, bool translate=false)	71
5.19.2.23	openProjectFile(const fs::path &path)	71
5.19.2.24	setTranslation(double tx, double ty, double tz)	71
5.19.2.25	update(PointCloud &p)	71
5.19.2.26	update(Room &r)	72

5.19.2.27	update(Floor &f)	72
5.19.2.28	update(Plane &p)	72
5.19.2.29	update(Wall &p)	72
5.19.2.30	update(Opening &p)	72
5.20	PTXWriter Class Reference	73
5.20.1	Member Function Documentation	73
5.20.1.1	write(std::ofstream &file, std::vector< PointXYZ > &points)	73
5.20.1.2	write(std::ofstream &file, std::vector< PointXYZNormal > &points)	73
5.21	Room Class Reference	74
5.21.1	Member Function Documentation	75
5.21.1.1	addPlane(Plane &pc)	75
5.21.1.2	addPointCloud(PointCloud &pc)	75
5.21.1.3	getArea() const	75
5.21.1.4	getCeilingLevel() const	75
5.21.1.5	getFloorLevel() const	75
5.21.1.6	getId() const	76
5.21.1.7	getMax() const	76
5.21.1.8	getMin() const	76
5.21.1.9	getPlaneIDs()	76
5.21.1.10	getPlanes()	76
5.21.1.11	getPlanes() const	76
5.21.1.12	getPointCloudIDs()	77
5.21.1.13	getPointClouds() const	77
5.21.1.14	getPointClouds()	77
5.21.1.15	getPoints(std::vector< T > &points)	77
5.21.1.16	setArea(double area)	77
5.21.1.17	setCeilingLevel(double l)	78
5.21.1.18	setFloorLevel(double l)	78
5.21.1.19	setId(double id)	78
5.21.1.20	setMax(double x, double y, double z)	78

5.21.1.21 setMin(double x, double y, double z)	78
5.21.1.22 setPlaneIDs(std::vector< int > pcl_ids)	79
5.21.1.23 setPointCloudIDs(std::vector< int > pcl_ids)	79
5.22 Vector3 Class Reference	79
5.22.1 Constructor & Destructor Documentation	80
5.22.1.1 Vector3(double vx, double vy, double vz)	80
5.22.1.2 Vector3(double vx, double vy)	80
5.22.1.3 Vector3(PointXYZ &p)	81
5.22.1.4 Vector3(PointXYZNormal &p)	81
5.22.2 Member Function Documentation	81
5.22.2.1 length()	81
5.22.2.2 length_sqr()	81
5.22.2.3 normalize()	81
5.22.2.4 normalized()	82
5.22.2.5 operator*(const double s)	82
5.22.2.6 operator*(const Vector3 o)	82
5.22.2.7 operator*=(const double s)	82
5.22.2.8 operator+(const Vector3 &o)	83
5.22.2.9 operator+=(const Vector3 &o)	83
5.22.2.10 operator-()	83
5.22.2.11 operator-(const Vector3 o)	83
5.22.2.12 operator-=(const Vector3 o)	84
5.22.2.13 operator/(const double s)	84
5.22.2.14 operator/=(const double s)	84
5.22.2.15 operator^(const Vector3 o)	84
5.22.2.16 operator^=(const Vector3 o)	85
5.23 Wall Class Reference	85
5.23.1 Member Function Documentation	86
5.23.1.1 addPlane(Plane &pc)	86
5.23.1.2 addRoom(Room &room)	86

5.23.1.3	<code>getFloorId() const</code>	86
5.23.1.4	<code>getId() const</code>	86
5.23.1.5	<code>getPlaneIDs()</code>	87
5.23.1.6	<code>getPlanes() const</code>	87
5.23.1.7	<code>getPlanes()</code>	87
5.23.1.8	<code>getPoints(std::vector< T > &points)</code>	87
5.23.1.9	<code>getRoomIDs()</code>	87
5.23.1.10	<code>getRooms() const</code>	88
5.23.1.11	<code>getRooms()</code>	88
5.23.1.12	<code>isOuter() const</code>	88
5.23.1.13	<code>setFloorId(int id)</code>	88
5.23.1.14	<code>setId(int id)</code>	88
5.23.1.15	<code>setOuter(bool outer)</code>	88
5.23.1.16	<code>setPlaneIDs(std::vector< int > pcl_ids)</code>	89
5.23.1.17	<code>setRoomIDs(std::vector< int > pcl_ids)</code>	89
6	File Documentation	91
6.1	<code>/home/edita/spcl2/src/filesystem_utils.h</code> File Reference	91
6.1.1	Detailed Description	91
6.1.2	Function Documentation	91
6.1.2.1	<code>directory_exists(const fs::path &p, fs::file_status s=fs::file_status{})</code>	91
6.2	<code>/home/edita/spcl2/src/MathUtils.h</code> File Reference	92
6.2.1	Detailed Description	94
6.2.2	Function Documentation	94
6.2.2.1	<code>angleBetweenPlanes(double a1, double b1, double c1, double a2, double b2, double c2)</code>	94
6.2.2.2	<code>angleBetweenPlanes(double n1[3], double n2[3])</code>	94
6.2.2.3	<code>averageD(std::vector< T > &points, double(&v1)[3], double(&averageD))</code>	95
6.2.2.4	<code>cross(double(&a)[3], double(&b)[3], double(&result)[3])</code>	95
6.2.2.5	<code>cubicEq(double au, double bu, double cu, double du, double(&result)[3])</code>	95
6.2.2.6	<code>det2(double a1, double b1, double c1, double d1)</code>	95

6.2.2.7	det3(double matrix[3][3])	96
6.2.2.8	dot(double ax, double ay, double az, double bx, double by, double bz)	96
6.2.2.9	eig3(double matrix[3][3], double lambda, double(&result)[3])	96
6.2.2.10	giveAverageCoord(std::vector< T > &points, int cIndex)	97
6.2.2.11	giveLocalCS(std::vector< T > &points, double(&locx)[3], double(&locy)[3], double(&locz)[3])	97
6.2.2.12	giveLtoGVectorsFromLocalCS(double(&locx)[3], double(&locy)[3], double(&locz)[3], double(&l2g1)[3], double(&l2g2)[3], double(&l2g3)[3])	97
6.2.2.13	moveBackCSorigin(std::vector< T > &points, T O)	97
6.2.2.14	moveCSorigin(std::vector< T > &points, T O)	98
6.2.2.15	normalize(double(&vec)[3])	98
6.2.2.16	planeFromPoints(std::vector< T > &points, std::vector< int > &indexes, double(&v1)[3], double &averagedD)	98
6.2.2.17	planeFromPoints(std::vector< T > &points, double(&v1)[3], double(&v2)[3], double(&v3)[3], double &averagedD)	98
6.2.2.18	projectPointsToXYPlane(std::vector< T > &points)	99
6.2.2.19	rotatePoint(T &pt, double(&v1)[3], double(&v2)[3], double(&v3)[3])	99
6.2.2.20	rotatePoints(std::vector< T > &points, double(&v1)[3], double(&v2)[3], double(&v3)[3])	99
6.2.2.21	rotatePointsToLocal(std::vector< T > &points, double(&locx)[3], double(&locy)[3])	99
6.3	/home/edita/spcl2/src/RANSAC.h File Reference	100
6.3.1	Detailed Description	100
6.3.2	Function Documentation	100
6.3.2.1	findPlanes(std::vector< T > &dataset, std::vector< std::vector< T >> &output)	100
6.3.2.2	plane_parameters(double(&point_coord)[3][3], double(&plane)[4])	101
6.3.2.3	random_three_pnt(std::vector< T > &data, double(&ThreePoint)[3][3], double thres)	101
6.4	/home/edita/spcl2/src/RegionGrowing.h File Reference	101
6.4.1	Detailed Description	102
6.4.2	Function Documentation	102
6.4.2.1	boundaryGrowing2(BitGrid &intermedPts, std::vector< BitGrid > &boundaries, std::vector< std::vector< int >> &indexes, std::vector< double > minIntermedPts)	102
6.4.2.2	getTrials(BitGrid &g, std::vector< int > &trials, int index)	103
6.4.2.3	getTrials2(BitGrid &g, std::vector< int > &trials, int index)	103
6.4.2.4	grow(BitGrid &g, std::vector< int > &room, int index)	103
6.4.2.5	inverseGrow(BitGrid &g, std::vector< int > &segment, int index)	104
6.4.2.6	inverseRegionGrowing(BitGrid &initialGrid, std::vector< BitGrid > &segList, std::vector< BitGrid > &boundaries, double minArea)	104
6.4.2.7	regionGrowing(BitGrid &g, std::vector< BitGrid > &segList, double minArea, int cellStep=1)	104
6.4.2.8	regionGrowingAround(BitGrid &g, std::vector< std::vector< int >> trialsVec, std::vector< BitGrid > &segList)	105

Chapter 1

PointCloud2BIM Library reference manual

The 3D laser scanning is a convenient and accessible technology to document the existing infrastructure including buildings. The output is set of data points in space, so called point clouds. While point clouds can be directly rendered and inspected, they need to be converted into BIM representation for efficient digital processing.

The PointCloud2BIM library provides a set of algorithms to facilitate the processing of Building Point Clouds and identification of fundamental entities, such as floors, rooms, walls and openings.

The PointCloud2BIM library is written in C++. It can be integrated into any BIM software to assist the conversion of point clouds into digital BIM representation.

This reference manual documents the API of the library.

Note, that the library can also be used as a standalone tool without any dependency on BIM software by offering a set of command line tools that can perform the typical steps in the transformation from the point clouds to BIM representation.

The library can be obtained from Department of Mechanics, Faculty of Civil Engineering, Czech Technical University. Please contact Borek Patzak (borek.patzak@fsv.cvut.cz) for details.

Copyright © 2020 by E. Dvořáková, B. Patzák, D. Rypl and J. Voříšek.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

BitGrid	9
BoundingBoxInfo	16
Cell	18
Downsampler	19
Floor	23
Histogram	28
HistogramInfo	30
Opening	31
Plane	35
Point< Dimensions >	46
Point< 2 >	46
PointXY	55
Point< 3 >	46
PointXYZ	58
PointXYZNormal	61
PointCloud	47
PointNormal	52
PointXYZNormal	61
PointXYZInfo	61
PointXYZNormalInfo	63
Polygon	63
Project	64
PTXWriter	73
Room	74
Vector3	79
Wall	85

Chapter 3

Class Index

3.1 Class List

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

BitGrid	9
BoundingBoxInfo	16
Cell	18
Downsampler	19
Floor	23
Histogram	28
HistogramInfo	30
Opening	31
Plane	35
Point< Dimensions >	46
PointCloud	47
PointNormal	52
PointXY	55
PointXYZ	58
PointXYZInfo	61
PointXYZNormal	61
PointXYZNormalInfo	63
Polygon	63
Project	64
PTXWriter	73
Room	74
Vector3	79
Wall	85

Chapter 4

File Index

4.1 File List

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

/home/edita/spcl2/src/filesystem_utils.h	
Set of functions supporting filesystem operations	91
/home/edita/spcl2/src/MathUtils.h	
Set of functions supporting mathematic operations	92
/home/edita/spcl2/src/RANSAC.h	
Set of functions suporting RANSAC algorithm	100
/home/edita/spcl2/src/RegionGrowing.h	
Set of functions for region growing	101

Chapter 5

Class Documentation

5.1 BitGrid Class Reference

Public Member Functions

- **BitGrid** (int dimensions, struct **BoundingBoxInfo** &bbox, double step)
Creates bitgrid with given dimensions.
- void **reset** ()
Reset vector of points and resize it to current size.
- const std::vector< bool > &**getPoints** ()
Get the points object.
- void **getActiveCellsIndex** (std::vector< int > &res)
Put active index ids into a vector.
- **BoundingBoxInfo** & **getBBox** ()
Get the bounding box struct.
- bool **get** (int i)
Get grid item by coordinate index (1d or nth point in vector).
- bool **get** (int x, int y)
Get grid item by coordinate indexes (2d).
- bool **get** (int x, int y, int z)
Get grid item by coordinate indexes (3d).
- int **getX** (int i)
Get x index coordinate from single vector index.
- int **getY** (int i)
Get y index coordinate from single vector index.
- bool **pointExists** (**PointXYZ** &p)
Check if points exists in our bit grid and if it is set true.
- **PointXYZ** **getPoint** (int i)
Return a point from vector index.
- double **getArea** ()
Get the area of our grid. Only works for 2d grids.
- int **getActiveCells** ()
Get the number of active cells (marked as true).
- int **getDimensions** ()
Get grid dimensions.
- int **getSize** ()

- Get the size of our grid (length of vector).*

 - int [getSizeX](#) ()
 - Get grid size in x direction.*
 - int [getSizeY](#) ()
 - Get grid size in y direction.*
 - int [getSizeZ](#) ()
 - Get grid size in z direction.*
 - double [getStepX](#) ()
 - Get size of grid step in the x direction.*
 - double [getStepY](#) ()
 - Get size of grid step in the y direction.*
 - double [getStepZ](#) ()
 - Get size of grid step in the z direction.*
 - void [invert](#) ()
 - Activate deactivated cells and deactivate activated. Used in backfill algorithm.*
 - void [backfill](#) ()
 - Backfill holes in the activated grid. Useful for found rooms not to miss some inner points.*
 - bool [add](#) ([PointXYZNormal](#) &p)
 - Activate grid cell. Cell coordinates are computed for the given point.*
 - bool [add](#) ([PointXYZ](#) &p)
 - Activate grid cell. Cell coordinates are computed for the given point.*
 - void [add](#) (int i)
 - Activate grid cell by 1d index.*
 - void [add](#) (int x, int y)
 - Activate grid by integer x,y coordinates.*
 - void [remove](#) (int i)
 - Deactivate grid cell by 1d index.*
 - std::vector< [PointXYZ](#) > [getBoundingPolygon](#) ()
 - Returns bounding polygon.*
 - svg::Document [toSVG](#) (std::filesystem::path path)
 - Saves [BitGrid](#) to SVG format.*

5.1.1 Constructor & Destructor Documentation

5.1.1.1 [BitGrid::BitGrid](#) (int *dimensions*, struct [BoundingBoxInfo](#) & *bbox*, double *step*)

Creates bitgrid with given dimensions.

Parameters

<i>dimensions</i>	Number of dimensions
<i>bbox</i>	Bounding box of BitGrid to be created
<i>step</i>	Cell size

5.1.2 Member Function Documentation

5.1.2.1 `bool BitGrid::add (PointXYZNormal & p)`

Activate grid cell. Cell coordinates are computed for the given point.

Parameters

<i>p</i>	Point with X,Y,Z coordinates
----------	------------------------------

Returns

True if cell was activated (index within grid bounds)
False if calculated grid index is out of bounds

5.1.2.2 `bool BitGrid::add (PointXYZ & p)`

Activate grid cell. Cell coordinates are computed for the given point.

Parameters

<i>p</i>	Point with X,Y,Z coordinates
----------	------------------------------

Returns

True if cell was activated (index within grid bounds)
False if calculated grid index is out of bounds

5.1.2.3 `void BitGrid::add (int i)`

Activate grid cell by 1d index.

Parameters

<i>i</i>	Index of a cell to be activated
----------	---------------------------------

5.1.2.4 `void BitGrid::add (int x, int y)`

Activate grid by integer x,y coordinates.

Parameters

<i>x</i>	X grid coordinate
<i>y</i>	Y grid coordinate

5.1.2.5 `bool BitGrid::get (int i)`

Get grid item by coordinate index (1d or nth point in vector).

Parameters

<i>i</i>	Index in vector
----------	-----------------

Returns

True if index exists and is marked true
False if index doesnt exist or is marked false

5.1.2.6 `bool BitGrid::get (int x, int y)`

Get grid item by coordinate indexes (2d).

Parameters

<i>x</i>	X index
<i>y</i>	Y index

Returns

True if index exists and is marked true
False if index doesnt exist or is marked false

5.1.2.7 `bool BitGrid::get (int x, int y, int z)`

Get grid item by coordinate indexes (3d).

Parameters

<i>x</i>	X index
<i>y</i>	Y index
<i>z</i>	Z index

Returns

True if indexes exist and are marked true
False if indexes do not exist or are marked false

5.1.2.8 `int BitGrid::getActiveCells ()`

Get the number of active cells (marked as true).

Returns

Number of active cells

5.1.2.9 void BitGrid::getActiveCellsIndex (std::vector< int > & res)

Put active index ids into a vector.

Parameters

<i>res</i>	Return parametr storing active cell indexes
------------	---

5.1.2.10 double BitGrid::getArea ()

Get the area of our grid. Only works for 2d grids.

Returns

Area in units squared

5.1.2.11 BoundingBoxInfo & BitGrid::getBBox ()

Get the bounding box struct.

Returns

[BoundingBoxInfo](#)

5.1.2.12 std::vector< PointXYZ > BitGrid::getBoundingPolygon ()

Returns bounding polygon.

Returns

Bounding polygon

5.1.2.13 int BitGrid::getDimensions ()

Get grid dimensions.

Returns

1 for 1d, 2 for 2d, 3 for 3d

5.1.2.14 PointXYZ BitGrid::getPoint (int i)

Return a point from vector index.

Parameters

<i>i</i>	Index of a cell
----------	-----------------

Returns

[PointXYZ](#)

5.1.2.15 `const std::vector< bool > & BitGrid::getPoints ()`

Get the points object.

Returns

Points object

5.1.2.16 `int BitGrid::getSize ()`

Get the size of our grid (length of vector).

Returns

Number of cells

5.1.2.17 `int BitGrid::getSizeX ()`

Get grid size in x direction.

Returns

Number of cells in the x direction

5.1.2.18 `int BitGrid::getSizeY ()`

Get grid size in y direction.

Returns

Number of cells in the y direction

5.1.2.19 `int BitGrid::getSizeZ ()`

Get grid size in z direction.

Returns

Number of cells in the z direction

5.1.2.20 double BitGrid::getStepX ()

Get size of grid step in the x direction.

Returns

Size of the x step

5.1.2.21 double BitGrid::getStepY ()

Get size of grid step in the y direction.

Returns

Size of the y step

5.1.2.22 double BitGrid::getStepZ ()

Get size of grid step in the z direction.

Returns

Size of the z step

5.1.2.23 int BitGrid::getX (int *i*)

Get x index coordinate from single vector index.

Parameters

<i>i</i>	Index of the cell
----------	-------------------

Returns

X coordinate

5.1.2.24 int BitGrid::getY (int *i*)

Get y index coordinate from single vector index.

Parameters

<i>i</i>	Index of the cell
----------	-------------------

Returns

Y coordinate

5.1.2.25 `bool BitGrid::pointExists (PointXYZ & p)`

Check if points exists in our bit grid and if it is set true.

Parameters

<i>p</i>	Point to check
----------	----------------

Returns

True if point is within grid bounds and flag is set to true
False if point doesn't exist in the grid or flag is se to false

5.1.2.26 `void BitGrid::remove (int i)`

Deactivate grid cell by 1d index.

Parameters

<i>i</i>	Index of a cell to be deactivated
----------	-----------------------------------

5.1.2.27 `svg::Document BitGrid::toSVG (std::filesystem::path path)`

Saves [BitGrid](#) to SVG format.

Parameters

<i>path</i>	Path to file where to save the SVG
-------------	------------------------------------

Returns

SVG Document

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

- /home/edita/spcl2/src/BitGrid.h
- /home/edita/spcl2/src/BitGrid.cpp

5.2 BoundingBoxInfo Struct Reference

Public Member Functions

- double [range](#) (int index)

- Calculates range of the coordinate given by index.*

 - void [giveBoundingBoxMin](#) (double(&min)[3])
 - Returns minimum coordinates of the [BoundingBoxInfo](#) object.*
 - void [giveBoundingBoxMax](#) (double(&max)[3])
 - Returns maximum coordinates of the [BoundingBoxInfo](#) object.*
 - [BoundingBoxInfo](#) ()
 - Construct a new [BoundingBoxInfo](#) object with infinities as bounds.*

Public Attributes

- double [min](#) [3]
 - Minimum coordinates [x, y, z] of the bounding box.*
- double [max](#) [3]
 - Maximum coordinates [x, y, z] of the bounding box.*

5.2.1 Member Function Documentation

5.2.1.1 void [BoundingBoxInfo::giveBoundingBoxMax](#) (double(& max[3])) `[inline]`

Returns maximum coordinates of the [BoundingBoxInfo](#) object.

Parameters

<i>max</i>	Maximum coordinates [x, y, z]
------------	-------------------------------

5.2.1.2 void [BoundingBoxInfo::giveBoundingBoxMin](#) (double(& min[3])) `[inline]`

Returns minimum coordinates of the [BoundingBoxInfo](#) object.

Parameters

<i>min</i>	Minimum coordinates [x, y, z]
------------	-------------------------------

5.2.1.3 double [BoundingBoxInfo::range](#) (int *index*) `[inline]`

Calculates range of the coordinate given by index.

Parameters

<i>index</i>	Specifies for which component (0-x, 1-y, 2-z) to calculate the range
--------------	--

Returns

Max coordinate - min coordinate

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

- /home/edita/spcl2/src/BoundingBox.h

5.3 Cell Struct Reference

Public Member Functions

- bool `operator==` (const `Cell` &o) const
Operator to compare two cells by index.

Public Attributes

- int `index`
Cell index.
- double `x` = 0.0
X coordinate of the cell (weighted average of all points)
- double `y` = 0.0
Y coordinate of the cell (weighted average of all points)
- double `z` = 0.0
Z coordinate of the cell (weighted average of all points)
- double `normal` [3]
Calculated normal vector of a cell.
- int `n` = 0
Number of points inside the cell.

Friends

- `size_t hash_value` (const `Cell` &p)
Hash `Cell` for usage inside of a hash map.

5.3.1 Member Function Documentation

5.3.1.1 `bool Cell::operator== (const Cell & o) const` [inline]

Operator to compare two cells by index.

Parameters

<code>o</code>	<code>Cell</code>
----------------	-------------------

Returns

True if cells are identical
False if cells indexes doesn't match

5.3.2 Friends And Related Function Documentation

5.3.2.1 `size_t hash_value (const Cell & p) [friend]`

Hash [Cell](#) for usage inside of a hash map.

Parameters

<code>p</code>	Cell to hash
----------------	--------------

Returns

Hash

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

- `/home/edita/spcl2/src/Downsampler.h`

5.4 Downsampler Class Reference

Public Member Functions

- [Downsampler](#) (double step)
Constructs downsampler with equal step size in all directions.
- [Downsampler](#) (double x, double y, double z)
Construct downsampler with given step sizes in x, y, z directions.
- void [setStep](#) (double step)
Sets equal step size in all direction.
- void [setStep](#) (double x, double y, double z)
Sets step sizes in x, y, z directions.
- int [getIndex](#) (int x, int y, int z)
Returns location index given by x, y, z position.
- int [getSize](#) ()
Returns map size.
- void [setSize](#) ([BoundingBoxInfo](#) &bb)
Sets sizes (in x, y, z direction) based on the given bounding box.
- bool [isActive](#) (int i)
Decides whether the cell is active or not.
- void [addPoint](#) ([PointXYZ](#) &p)
Adds given point to the map.
- void [downsample](#) ([BoundingBoxInfo](#) &bb, `std::vector< PointXYZ >` &points)
Adds given points to the map specified by the bounding box.
- void [getPoints](#) (`std::vector< PointXYZ >` &pts)
Returns points (with x, y, z coordinates) stored in the map.
- void [getPoints](#) (`std::vector< PointXYZNormal >` &pts)
Returns points (with x, y, z coordinates and normal vector) stored in the map.
- void [computeNormals](#) ()
Calculates normals.

5.4.1 Constructor & Destructor Documentation

5.4.1.1 Downsampler::Downsampler (double *step*) [*inline*]

Constructs downsampler with equal step size in all directions.

Parameters

<i>step</i>	Step to be set
-------------	----------------

5.4.1.2 Downsamplers::Downsamplers (double *x*, double *y*, double *z*) [inline]

Construct downsampler with given step sizes in x, y, z directions.

Parameters

<i>x</i>	Step to be set in x-direction
<i>y</i>	Step to be set in y-direction
<i>z</i>	Step to be set in z-direction

5.4.2 Member Function Documentation

5.4.2.1 void Downsamplers::addPoint (PointXYZ & *p*) [inline]

Adds given point to the map.

Parameters

<i>p</i>	Point to be added
----------	-------------------

5.4.2.2 void Downsamplers::downsample (BoundingBoxInfo & *bb*, std::vector< PointXYZ > & *points*) [inline]

Adds given points to the map specified by the bounding box.

Parameters

<i>bb</i>	Bounding box defining the map size
<i>points</i>	Points to be added

5.4.2.3 int Downsamplers::getIndex (int *x*, int *y*, int *z*) [inline]

Returns location index given by x, y, z position.

Parameters

<i>x</i>	Position in x-direction
<i>y</i>	Position in y-direction
<i>z</i>	Position in z-direction

Returns

Calculated index

5.4.2.4 `void Downsampler::getPoints (std::vector< PointXYZ > & pts)` `[inline]`

Returns points (with x, y, z coordinates) stored in the map.

Parameters

<i>pts</i>	Return parametr storing the points
------------	------------------------------------

5.4.2.5 `void Downsampler::getPoints (std::vector< PointXYZNormal > & pts)` `[inline]`

Returns points (with x, y, z coordinates and normal vector) stored in the map.

Parameters

<i>pts</i>	Return parametr storing the points
------------	------------------------------------

5.4.2.6 `int Downsampler::getSize ()` `[inline]`

Returns map size.

Returns

Map size

5.4.2.7 `bool Downsampler::isActive (int i)` `[inline]`

Decides whether the cell is active or not.

Parameters

<i>i</i>	Cell index
----------	------------

Returns

True if cell is active
False if cell is inactive

5.4.2.8 `void Downsampler::setSize (BoundingBoxInfo & bb)` `[inline]`

Sets sizes (in x, y, z direction) based on the given bounding box.

Parameters

<i>bb</i>	Bounding box based on which to set the size
-----------	---

5.4.2.9 void DownSampler::setStep (double *step*) [inline]

Sets equal step size in all direction.

Parameters

<i>step</i>	Step to be set
-------------	----------------

5.4.2.10 void DownSampler::setStep (double *x*, double *y*, double *z*) [inline]

Sets step sizes in x, y, z directions.

Parameters

<i>x</i>	Step to be set in x-direction
<i>y</i>	Step to be set in y-direction
<i>z</i>	Step to be set in z-direction

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

- /home/edita/spcl2/src/DownSampler.h

5.5 Floor Class Reference

Public Member Functions

- [Floor](#) ()
Creates empty floor.
- void [setId](#) (int id)
Sets Floor identifier.
- int [getId](#) () const
Returns Floor identifier.
- const double * [getMin](#) () const
Return Floor minimum coordinates.
- void [setMin](#) (double x, double y, double z)
Sets minimum Floor coordinates.
- const double * [getMax](#) () const
Return Floor maximum coordinates.
- void [setMax](#) (double x, double y, double z)
Sets maximum Floor coordinates.

- void [setFloorLevel](#) (double l)
Sets floor level (height).
- double [getFloorLevel](#) () const
Returns floor level (height).
- void [setCeilingLevel](#) (double l)
Sets ceiling level (height).
- double [getCeilingLevel](#) () const
Returns ceiling level (height).
- std::vector< int > & [getRoomIDs](#) ()
Returns vector of [Room](#) identifiers.
- void [setRoomIDs](#) (std::vector< int > room_ids)
Set vector of [Room](#) identifiers.
- const std::vector< [Room](#) > & [getRooms](#) () const
Returns vector of [Rooms](#).
- std::vector< [Room](#) > & [getRooms](#) ()
Returns vector of [Rooms](#).
- void [addRoom](#) ([Room](#) &room)
Adds [Room](#) to [Floor](#).
- std::vector< int > & [getWallIDs](#) ()
Returns vector of [Wall](#) identifiers.
- void [setWallIDs](#) (std::vector< int > wall_ids)
Sets vector of [Wall](#) identifiers.
- const std::vector< [Wall](#) > & [getWalls](#) () const
Returns all [Walls](#).
- std::vector< [Wall](#) > & [getWalls](#) ()
Returns all [Walls](#).
- void [addWall](#) ([Wall](#) &wall)
Adds [Wall](#) to [Floor](#).
- template<typename T >
void [getPoints](#) (std::vector< T > &points)
Returns all [Floor](#) points.

5.5.1 Member Function Documentation

5.5.1.1 void Floor::addRoom ([Room](#) & room) [inline]

Adds [Room](#) to [Floor](#).

Parameters

<i>room</i>	Room to be added
-------------	----------------------------------

5.5.1.2 void Floor::addWall ([Wall](#) & wall) [inline]

Adds [Wall](#) to [Floor](#).

Parameters

<i>wall</i>	Wall
-------------	------

5.5.1.3 `double Floor::getCeilingLevel () const` [inline]

Returns ceiling level (height).

Returns

Ceiling level

5.5.1.4 `double Floor::getFloorLevel () const` [inline]

Returns floor level (height).

Returns

Floor level

5.5.1.5 `int Floor::getId () const` [inline]

Returns Floor identifier.

Returns

Floor identifier

5.5.1.6 `const double* Floor::getMax () const` [inline]

Return Floor maximum coordinates.

Returns

Array of maximum Floor coordinates

5.5.1.7 `const double* Floor::getMin () const` [inline]

Return Floor minimum coordinates.

Returns

Array of minimum Floor coordinates

5.5.1.8 `template<typename T > void Floor::getPoints (std::vector< T > & points)` [inline]

Returns all Floor points.

Template Parameters

<i>T</i>	Point type
----------	------------

Parameters

<i>points</i>	Points
---------------	--------

5.5.1.9 `std::vector<int>& Floor::getRoomIDs ()` [inline]

Returns vector of [Room](#) identifiers.

Returns

[Room](#) identifiers

5.5.1.10 `const std::vector<Room>& Floor::getRooms () const` [inline]

Returns vector of Rooms.

Returns

Rooms

5.5.1.11 `std::vector<Room>& Floor::getRooms ()` [inline]

Returns vector of Rooms.

Returns

Rooms

5.5.1.12 `std::vector<int>& Floor::getWallIDs ()` [inline]

Returns vector of [Wall](#) identifiers.

Returns

[Wall](#) identifiers

5.5.1.13 `const std::vector<Wall>& Floor::getWalls () const` [inline]

Returns all Walls.

Returns

Walls

5.5.1.14 `std::vector<Wall>& Floor::getWalls ()` [inline]

Returns all Walls.

Returns

Walls

5.5.1.15 `void Floor::setCeilingLevel (double l)` [inline]

Sets ceiling level (height).

Parameters

<i>l</i>	Ceiling level
----------	---------------

5.5.1.16 `void Floor::setFloorLevel (double l)` [inline]

Sets floor level (height).

Parameters

<i>l</i>	Floor level
----------	-------------

5.5.1.17 `void Floor::setId (int id)` [inline]

Sets Floor identifier.

Parameters

<i>id</i>	Floor identifier to be set
-----------	----------------------------

5.5.1.18 `void Floor::setMax (double x, double y, double z)` [inline]

Sets maximum Floor coordinates.

Parameters

<i>x</i>	Maximal x coordinate
<i>y</i>	Maximal y coordinate
<i>z</i>	Maximal z coordinate

5.5.1.19 void Floor::setMin (double x, double y, double z) [inline]

Sets minimum [Floor](#) coordinates.

Parameters

x	Minimal x coordinate
y	Minimal y coordinate
z	Minimal z coordinate

5.5.1.20 void Floor::setRoomIDs (std::vector< int > room_ids) [inline]

Set vector of [Room](#) identifiers.

Parameters

room_ids	Room identifiers
----------	----------------------------------

5.5.1.21 void Floor::setWallIDs (std::vector< int > wall_ids) [inline]

Sets vector of [Wall](#) identifiers.

Parameters

wall_ids	Wall identifiers
----------	----------------------------------

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

- /home/edita/spcl2/src/Floor.h

5.6 Histogram Class Reference

Public Member Functions

- [Histogram](#) ()
Constructs empty histogram.
- [Histogram](#) ([BoundingBoxInfo](#) &bbox, double step, int coord)
Constructs histogram in given bounding box with step size in given direction.
- void [addPoint](#) ([PointXYZ](#) &p)
Adds point to the histogram.
- void [addPoint](#) ([PointXYZNormal](#) &p)
Adds point to the histogram.
- std::vector< [HistogramInfo](#) > [getHistogram](#) ()
Returns [HistogramInfo](#).

Static Public Member Functions

- static `std::vector< HistogramInfo > computeHistogram (std::vector< PointXYZ > &points, BoundingBoxInfo &bbox, double step, int coord)`

Calculates histogram.

5.6.1 Constructor & Destructor Documentation

5.6.1.1 Histogram::Histogram (BoundingBoxInfo & *bbox*, double *step*, int *coord*) `[inline]`

Constructs histogram in given bounding box with step size in given direction.

Parameters

<i>bbox</i>	Bounding box specifying the histogram ranges
<i>step</i>	Step size
<i>coord</i>	Coordinate for which to calculate the histogram

5.6.2 Member Function Documentation

5.6.2.1 void Histogram::addPoint (PointXYZ & *p*) `[inline]`

Adds point to the histogram.

Parameters

<i>p</i>	Point to be added
----------	-----------------------------------

5.6.2.2 void Histogram::addPoint (PointXYZNormal & *p*) `[inline]`

Adds point to the histogram.

Parameters

<i>p</i>	Point to be added
----------	-----------------------------------

5.6.2.3 static `std::vector<HistogramInfo> Histogram::computeHistogram (std::vector< PointXYZ > & points, BoundingBoxInfo & bbox, double step, int coord) [inline], [static]`

Calculates histogram.

Parameters

<i>points</i>	Points for histogram calculation
---------------	----------------------------------

Parameters

<i>bbox</i>	Calculation bounding box
<i>step</i>	Histogram step
<i>coord</i>	Histogram direction

Returns

[HistogramInfo](#)

5.6.2.4 `std::vector<HistogramInfo> Histogram::getHistogram ()` [*inline*]

Returns [HistogramInfo](#).

Returns

[HistogramInfo](#)

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

- /home/edita/spcl2/src/Histogram.h

5.7 HistogramInfo Struct Reference

Public Attributes

- int [index](#) = 0
Order of histogram step (to be preserved while sorting)
- double [coordMin](#) = 0
Starting histogram step coordinate.
- double [coordMax](#) = 0
Ending histogram step coordinate.
- int [n](#) = 0
Number of points inside the histogram step.
- double [relative](#) = 0.0
Relative percentage of points in the histogram step.

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

- /home/edita/spcl2/src/Histogram.h

5.8 Opening Class Reference

Public Member Functions

- [Opening](#) ()
Creates empty [Opening](#).
- int [getId](#) () const
Returns [Opening](#) identifier.
- void [setId](#) (int id)
Sets [Opening](#) identifier.
- std::vector< int > & [getPointCloudIDs](#) ()
Returns vector of [PointCloud](#) identifiers.
- void [setPointCloudIDs](#) (std::vector< int > pcl_ids)
Sets vector of [PointCloud](#) identifiers.
- const std::vector< [PointCloud](#) > & [getPointClouds](#) () const
Get all [Opening](#) pointclouds.
- std::vector< [PointCloud](#) > & [getPointClouds](#) ()
Get all [Opening](#) pointclouds.
- void [addPointCloud](#) ([PointCloud](#) &pc)
Adds [PointCloud](#) to [Opening](#).
- void [setDimensions](#) (std::vector< double > dimensions)
Sets dimensions vector.
- const std::vector< double > & [getDimensions](#) () const
Returns dimensions vector.
- std::vector< double > & [getDimensions](#) ()
Returns dimensions vector.
- void [setCornersX](#) (std::vector< double > cornersX)
Sets x coordinates of the points (corners) specifying [Opening](#).
- const std::vector< double > & [getCornersX](#) () const
Returns x coordinates of the points (corners) specifying [Opening](#).
- void [setCornersY](#) (std::vector< double > cornersY)
Sets y coordinates of the points (corners) specifying [Opening](#).
- const std::vector< double > & [getCornersY](#) () const
Returns x coordinates of the points (corners) specifying [Opening](#).
- void [setCornersZ](#) (std::vector< double > cornersZ)
Sets z coordinates of the points (corners) specifying [Opening](#).
- const std::vector< double > & [getCornersZ](#) () const
Returns z coordinates of the points (corners) specifying [Opening](#).
- OpeningType [getOpeningType](#) () const
Returns type of the opening.
- void [setOpeningType](#) (OpeningType name)
Sets type of the opening.
- template<typename T >
void [getPoints](#) (std::vector< T > &points)
Returns all [Opening](#) points.

5.8.1 Member Function Documentation

5.8.1.1 void [Opening::addPointCloud](#) ([PointCloud](#) & pc) [`inline`]

Adds [PointCloud](#) to [Opening](#).

Parameters

<i>pc</i>	PointCloud
-----------	----------------------------

5.8.1.2 `const std::vector<double>& Opening::getCornersX () const` `[inline]`

Returns x coordinates of the points (corners) specifying [Opening](#).

Returns

X coordinates of the corner points

5.8.1.3 `const std::vector<double>& Opening::getCornersY () const` `[inline]`

Returns x coordinates of the points (corners) specifying [Opening](#).

Returns

Y coordinates of the corner points

5.8.1.4 `const std::vector<double>& Opening::getCornersZ () const` `[inline]`

Returns z coordinates of the points (corners) specifying [Opening](#).

Returns

Z coordinates of the corner points

5.8.1.5 `const std::vector<double>& Opening::getDimensions () const` `[inline]`

Returns dimensions vector.

Returns

Dimensions

5.8.1.6 `std::vector<double>& Opening::getDimensions ()` `[inline]`

Returns dimensions vector.

Returns

Dimensions

5.8.1.7 `int Opening::getId () const [inline]`

Returns [Opening](#) identifier.

Returns

[Opening](#) identifier

5.8.1.8 `OpeningType Opening::getOpeningType () const [inline]`

Returns type of the opening.

Returns

[Opening](#) type

5.8.1.9 `std::vector<int>& Opening::getPointCloudIDs () [inline]`

Returns vector of [PointCloud](#) identifiers.

Returns

[PointCloud](#) identifiers

5.8.1.10 `const std::vector<PointCloud>& Opening::getPointClouds () const [inline]`

Get all [Opening](#) pointclouds.

Returns

PointClouds

5.8.1.11 `std::vector<PointCloud>& Opening::getPointClouds () [inline]`

Get all [Opening](#) pointclouds.

Returns

PointClouds

5.8.1.12 `template<typename T> void Opening::getPoints (std::vector< T > & points) [inline]`

Returns all [Opening](#) points.

Template Parameters

<i>T</i>	Point type
----------	------------

Parameters

<i>points</i>	Points
---------------	--------

5.8.1.13 void `Opening::setCornersX (std::vector< double > cornersX)` `[inline]`

Sets x coordinates of the points (corners) specifying [Opening](#).

Parameters

<i>cornersX</i>	X coordinates of the corner points
-----------------	------------------------------------

5.8.1.14 void `Opening::setCornersY (std::vector< double > cornersY)` `[inline]`

Sets y coordinates of the points (corners) specifying [Opening](#).

Parameters

<i>cornersY</i>	Y coordinates of the corner points
-----------------	------------------------------------

5.8.1.15 void `Opening::setCornersZ (std::vector< double > cornersZ)` `[inline]`

Sets z coordinates of the points (corners) specifying [Opening](#).

Parameters

<i>cornersZ</i>	Z coordinates of the corner points
-----------------	------------------------------------

5.8.1.16 void `Opening::setDimensions (std::vector< double > dimensions)` `[inline]`

Sets dimensions vector.

Parameters

<i>dimensions</i>	Dimensions
-------------------	------------

5.8.1.17 void Opening::setld (int *id*) [inline]

Sets [Opening](#) identifier.

Parameters

<i>id</i>	Opening identifier
-----------	------------------------------------

5.8.1.18 void Opening::setOpeningType (OpeningType *name*) [inline]

Sets type of the opening.

Parameters

<i>name</i>	Opening type
-------------	------------------------------

5.8.1.19 void Opening::setPointCloudIDs (std::vector< int > *pcl_ids*) [inline]

Sets vector of [PointCloud](#) identifiers.

Parameters

<i>pcl_ids</i>	PointCloud identifiers
----------------	--

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

- /home/edita/spcl2/src/Opening.h

5.9 Plane Class Reference

Public Member Functions

- [Plane](#) ()
Creates empty plane.
- [Plane](#) (double *_a*, double *_b*, double *_c*, double *_d*)
Creates plane from given plane coefficients ($ax + by + cz + d = 0$).
- [Plane](#) ([Vector3](#) *_a*, [Vector3](#) *_b*, [Vector3](#) *_c*)
Creates plane from the coordinates (x,y,z) of the three given points.
- [Plane](#) ([PointXYZ](#) *_a*, [PointXYZ](#) *_b*, [PointXYZ](#) *_c*)
Creates plane from three given points ([PointXYZNormal](#)).
- [Plane](#) ([PointXYZNormal](#) *_a*, [PointXYZNormal](#) *_b*, [PointXYZNormal](#) *_c*)
Creates plane from three given points ([PointXYZNormal](#)).
- const double * [getNormal](#) () const
Returns plane normal vector.

- double * [getNormal](#) ()
Returns plane normal vector.
- void [setNormal](#) (double nx, double ny, double nz)
Sets plane normal vector.
- const double * [getMin](#) () const
Returns minimum coordinates.
- void [setMin](#) (double x, double y, double z)
Sets minimum coordinates.
- const double * [getMax](#) () const
Returns maximum coordinates.
- void [setMax](#) (double x, double y, double z)
Sets maximum coordinates.
- int [getld](#) () const
Returns [Plane](#) identifier.
- void [setld](#) (int id)
Sets [Plane](#) identifier.
- int [getFloorld](#) () const
Returns [Floor](#) identifier.
- void [setFloorld](#) (int id)
Sets [Floor](#) identifier.
- double [getD](#) () const
Returns coefficient d ($Plane = ax + by + cz + d = 0$).
- void [setD](#) (double d)
Sets coefficient d ($Plane = ax + by + cz + d = 0$).
- std::vector< int > & [getPointCloudIDs](#) ()
Returns [Plane](#) PointClouds identifiers.
- void **setPointCloudIDs** (std::vector< int > pcl_ids)
- const std::vector< [PointCloud](#) > & [getPointClouds](#) () const
Returns [Plane](#) PointClouds.
- std::vector< [PointCloud](#) > & [getPointClouds](#) ()
Returns [Plane](#) PointClouds.
- void [addPointCloud](#) ([PointCloud](#) &pc)
Adds [PointCloud](#) to the [Plane](#).
- template<typename T >
void [getPoints](#) (std::vector< T > &points)
Returns [Plane](#) Points.
- [Vector3](#) [getNormalVector](#) ()
Returns [Plane](#) normal vector.
- double [distanceTo](#) ([Vector3](#) &v)
Returns the distance between the receiver and the given point.
- double [distanceTo](#) ([PointXYZ](#) &p)
Returns the distance between the receiver and the given point.
- double [distanceTo](#) ([PointXYZNormal](#) p)
Returns the distance between the receiver and the given point.
- double **distanceTo** (std::vector< [PointXYZ](#) > &points)
- double [distanceTo](#) (std::vector< [PointXYZNormal](#) > &points)
Returns the distance between the receiver and the given point.

Static Public Member Functions

- static double **angleBetween** (double a1, double b1, double c1, double a2, double b2, double c2)
- static void **getLocalContinuity** (std::vector< [PointXYZ](#) > points1, std::vector< [PointXYZ](#) > points2, std::vector< int > &result)
- static void **getLocalMatch** (std::vector< [PointXYZNormal](#) > points1, std::vector< [PointXYZNormal](#) > points2, std::vector< int > &result1, std::vector< int > &result2)

Search for the overlapping points of two given planes.
- static bool **makeWall** ([Plane](#) &plane1, [Plane](#) &plane2, std::vector< [PointXYZ](#) > &result1, std::vector< [PointXYZ](#) > &result2)

Search for the overlapping points of two given planes.
- static double **checkLocalMatch** (std::vector< [PointXYZNormal](#) > points1, std::vector< [PointXYZNormal](#) > points2)

Returns the ratio of the overlapping area of two planes over the area of the smaller plane.
- static double **checkMatchingArea** (std::vector< [PointXYZNormal](#) > points1, std::vector< [PointXYZNormal](#) > points2)

Returns the size of the overlapping area of two planes.
- static void **giveCenterPlane** (std::vector< [PointXYZNormal](#) > points1, std::vector< [PointXYZNormal](#) > points2, double(&v1)[3], double(&v2)[3], double(&v3)[3], double &d, std::vector< [PointXYZNormal](#) > &corners)

Calculates the centerplane between the two given planes.
- static bool **isInnerPlane** (std::vector< [PointXYZNormal](#) > points, std::vector< [BitGrid](#) > roomGrids, double minOffset)

Evaluates whether the plane is inner (between two rooms) or not (at the pointcloud boundary).
- static bool **generateOuterPlane** (std::vector< [PointXYZNormal](#) > points, std::vector< [PointXYZNormal](#) > &outPoints, std::vector< [BitGrid](#) > roomGrids, double minOffset)

Generates a copy of the plane at the pointcloud boundary with given offset.
- static double **distanceBetweenPlanes** ([Plane](#) &p1, [Plane](#) &p2)

Returns distance between planes with same a, b, c coefficients.

5.9.1 Constructor & Destructor Documentation

5.9.1.1 `Plane::Plane (double _a, double _b, double _c, double _d) [inline]`

Creates plane from given plane coefficients ($ax + by + cz + d = 0$).

Parameters

<code>↔</code> <code>↔</code> <code>a</code>	Coefficient a
<code>↔</code> <code>↔</code> <code>b</code>	Coefficient b
<code>↔</code> <code>↔</code> <code>c</code>	Coefficient c
<code>↔</code> <code>↔</code> <code>d</code>	Coefficient d

5.9.1.2 `Plane::Plane (Vector3_a, Vector3_b, Vector3_c) [inline]`

Creates plane from the coordinates (x,y,z) of the three given points.

Parameters

↔ _↔ a	Vector of the coordinates of the point 1
↔ _↔ b	Vector of the coordinates of the point 2
↔ _↔ c	Vector of the coordinates of the point 3

5.9.1.3 `Plane::Plane (PointXYZ_a, PointXYZ_b, PointXYZ_c) [inline]`

Creates plane from three given points ([PointXYZNormal](#)).

Parameters

↔ _↔ a	Point 1
↔ _↔ b	Point 2
↔ _↔ c	Point 3

5.9.1.4 `Plane::Plane (PointXYZNormal_a, PointXYZNormal_b, PointXYZNormal_c) [inline]`

Creates plane from three given points ([PointXYZNormal](#)).

Parameters

↔ _↔ a	Point 1
↔ _↔ b	Point 2
↔ _↔ c	Point 3

5.9.2 Member Function Documentation

5.9.2.1 void Plane::addPointCloud (PointCloud & *pc*) [inline]

Adds [PointCloud](#) to the [Plane](#).

Parameters

<i>pc</i>	PointCloud to be added
-----------	--

5.9.2.2 static double Plane::checkLocalMatch (std::vector< PointXYZNormal > *points1*, std::vector< PointXYZNormal > *points2*) [inline],[static]

Returns the ratio of the overlapping area of two planes over the area of the smaller plane.

Parameters

<i>points1</i>	Points of the first plane
<i>points2</i>	Points of the second plane

Returns

Matched percentage

5.9.2.3 static double Plane::checkMatchingArea (std::vector< PointXYZNormal > *points1*, std::vector< PointXYZNormal > *points2*) [inline],[static]

Returns the size of the overlapping area of two planes.

Parameters

<i>points1</i>	Points of the first plane
<i>points2</i>	Points of the second plane

Returns

Matching area

5.9.2.4 static double Plane::distanceBetweenPlanes (Plane & *p1*, Plane & *p2*) [inline],[static]

Returns distance between planes with same a, b, c coefficients.

Parameters

<i>p1</i>	The first plane
<i>p2</i>	The second plane

Returns

Calculated distance

5.9.2.5 `double Plane::distanceTo (Vector3 & v) [inline]`

Returns the distance between the receiver and the given point.

Parameters

<code>v</code>	Vector of the coordinates (x, y, z) of the point to which the distance is calculated
----------------	--

Returns

Distance of the point to the plane

5.9.2.6 `double Plane::distanceTo (PointXYZ & p) [inline]`

Returns the distance between the receiver and the given point.

Parameters

<code>p</code>	Point to which the distance is calculated
----------------	---

Returns

Distance of the point to the plane

5.9.2.7 `double Plane::distanceTo (PointXYZNormal p) [inline]`

Returns the distance between the receiver and the given point.

Parameters

<code>p</code>	Point to which the distance is calculated
----------------	---

Returns

Distance of the point to the plane

5.9.2.8 `double Plane::distanceTo (std::vector< PointXYZNormal > & points) [inline]`

Returns the distance between the receiver and the given point.

Parameters

p	Point to which the distance is calculated
-----	---

Returns

Distance of the point to the plane

5.9.2.9 `static bool Plane::generateOuterPlane (std::vector< PointXYZNormal > points, std::vector< PointXYZNormal > & outPoints, std::vector< BitGrid > roomGrids, double minOffset) [inline],[static]`

Generates a copy of the plane at the pointcloud boundary with given offset.

Parameters

<i>points</i>	Points of the plane to be copied
<i>outPoints</i>	Return parameter storing points of the newly generated plane
<i>roomGrids</i>	Vector of grids of all rooms in the pointcloud
<i>minOffset</i>	Parameter defining the offset (in the normal direction, positive value ~ out of the pointcloud) of the newly generated plane

Returns

True if plane is inner
False if plane is at the pointcloud boundary

5.9.2.10 `double Plane::getD () const [inline]`

Returns coefficient d ($\text{Plane} = ax + by + cz + d = 0$).

Returns

Coefficient d

5.9.2.11 `int Plane::getFloorId () const [inline]`

Returns Floor identifier.

Returns

Floor identifier

5.9.2.12 `int Plane::getId () const [inline]`

Returns Plane identifier.

Returns

Plane identifier

5.9.2.13 `static void Plane::getLocalMatch (std::vector< PointXYZNormal > points1, std::vector< PointXYZNormal > points2, std::vector< int > & result1, std::vector< int > & result2)` `[inline],[static]`

Search for the overlapping points of two given planes.

Parameters

<i>points1</i>	Points of the first plane
<i>points2</i>	Points of the second plane
<i>result1</i>	Indexes of the points1 which are overlapping with the second plane
<i>result2</i>	Indexes of the points2 which are overlapping with the first plane

5.9.2.14 `const double* Plane::getMax () const` `[inline]`

Returns maximum coordinates.

Returns

Maximum coordinates [x, y, z]

5.9.2.15 `const double* Plane::getMin () const` `[inline]`

Returns minimum coordinates.

Returns

Minimum coordinates [x, y, z]

5.9.2.16 `const double* Plane::getNormal () const` `[inline]`

Returns plane normal vector.

Returns

[Plane](#) normal vector.

5.9.2.17 `double* Plane::getNormal ()` `[inline]`

Returns plane normal vector.

Returns

[Plane](#) normal vector.

5.9.2.18 `Vector3 Plane::getNormalVector () [inline]`

Returns [Plane](#) normal vector.

Returns

Normal vector

5.9.2.19 `std::vector<int>& Plane::getPointCloudIDs () [inline]`

Returns [Plane](#) PointClouds identifiers.

Returns

PointClouds identifiers

5.9.2.20 `const std::vector<PointCloud>& Plane::getPointClouds () const [inline]`

Returns [Plane](#) PointClouds.

Returns

[Plane](#) PointClouds

5.9.2.21 `std::vector<PointCloud>& Plane::getPointClouds () [inline]`

Returns [Plane](#) PointClouds.

Returns

[Plane](#) PointClouds

5.9.2.22 `template<typename T> void Plane::getPoints (std::vector<T> & points) [inline]`

Returns [Plane](#) Points.

Parameters

<i>points</i>	Return parametr storing Plane points
---------------	--

5.9.2.23 `static void Plane::giveCenterPlane (std::vector< PointXYZNormal > points1, std::vector< PointXYZNormal > points2, double(&) v1[3], double(&) v2[3], double(&) v3[3], double & d, std::vector< PointXYZNormal > & corners) [inline],[static]`

Calculates the centerplane between the two given planes.

Parameters

<i>points1</i>	Plane 1
<i>points2</i>	Plane 2
<i>v1</i>	Return parameter corresponding to the normal vector of the calculated centerplane
<i>v2</i>	Return parameter corresponding to the in-plane vector of the calculated centerplane
<i>v3</i>	Return parameter corresponding to the in-plane vector of the calculated centerplane
<i>d</i>	Return parameter corresponding to coefficient d of the calculated plane
<i>corners</i>	Return parameter corresponding to the set of points defining the centerplane

5.9.2.24 `static bool Plane::isInnerPlane (std::vector< PointXYZNormal > points, std::vector< BitGrid > roomGrids, double minOffset) [inline],[static]`

Evaluates whether the plane is inner (between two rooms) or not (at the pointcloud boundary).

Parameters

<i>points</i>	Points of the plane to be evaluated
<i>roomGrids</i>	Vector of grids of all rooms in the pointcloud
<i>minOffset</i>	Parameter defining how far from the plane to search for the room

Returns

True if plane is inner
False if plane is at the pointcloud boundary

5.9.2.25 `static bool Plane::makeWall (Plane & plane1, Plane & plane2, std::vector< PointXYZ > & result1, std::vector< PointXYZ > & result2) [inline],[static]`

Search for the overlapping points of two given planes.

Parameters

<i>points1</i>	Points of the first plane
<i>points2</i>	Points of the second plane
<i>result1</i>	Return parametr storing indexes of the points1 which are overlapping with the second plane
<i>result2</i>	Return parametr storing indexes of the points2 which are overlapping with the first plane

5.9.2.26 `void Plane::setD (double d) [inline]`

Sets coefficient d (Plane = $ax + by + cz + d = 0$).

Parameters

<i>d</i>	Coefficient d to be set
----------	-------------------------

5.9.2.27 `void Plane::setFloorId (int id) [inline]`

Sets [Floor](#) identifier.

Parameters

<i>id</i>	Floor identifier to be set
-----------	--

5.9.2.28 `void Plane::setId (int id) [inline]`

Sets [Plane](#) identifier.

Parameters

<i>id</i>	Plane identifier to be set
-----------	--

5.9.2.29 `void Plane::setMax (double x, double y, double z) [inline]`

Sets maximum coordinates.

Parameters

<i>x</i>	X coordinate to be set
<i>y</i>	Y coordinate to be set
<i>z</i>	Z coordinate to be set

5.9.2.30 `void Plane::setMin (double x, double y, double z) [inline]`

Sets minimum coordinates.

Parameters

<i>x</i>	X coordinate to be set
<i>y</i>	Y coordinate to be set
<i>z</i>	Z coordinate to be set

5.9.2.31 `void Plane::setNormal (double nx, double ny, double nz) [inline]`

Sets plane normal vector.

Parameters

<i>nx</i>	X component
<i>ny</i>	Y component
<i>nz</i>	Z component

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

- /home/edita/spcl2/src/Plane.h

5.10 Point< Dimensions > Class Template Reference

Public Member Functions

- double * [getCoords](#) ()
Returns Point coordinates.
- virtual void [setId](#) (double id)
Sets Point identifier.
- virtual int [getId](#) ()
Returns Point identifier.

5.10.1 Member Function Documentation

5.10.1.1 `template<int Dimensions> double* Point< Dimensions >::getCoords () [inline]`

Returns [Point](#) coordinates.

Returns

[Point](#) coordinates

5.10.1.2 `template<int Dimensions> virtual int Point< Dimensions >::getId () [inline],[virtual]`

Returns [Point](#) identifier.

Returns

[Point](#) identifier

5.10.1.3 `template<int Dimensions> virtual void Point< Dimensions >::setId (double id) [inline],[virtual]`

Sets [Point](#) identifier.

Parameters

<i>id</i>	Point identifier to be set
-----------	--

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

- /home/edita/spcl2/src/Point.h

5.11 PointCloud Class Reference

Public Member Functions

- [PointCloud](#) ()=default
Constructs empty [PointCloud](#).
- [PointCloud](#) (std::string name)
Constructs [PointCloud](#) with given name.
- void [setId](#) (double id)
Sets [PointCloud](#) identifier.
- int [getId](#) () const
Returns [PointCloud](#) identifier.
- std::string [getName](#) () const
Returns [PointCloud](#) name.
- void [setName](#) (std::string name)
Sets [PointCloud](#) name.
- PointType [getPointType](#) () const
Returns [PointType](#).
- void [setPointType](#) (PointType name)
Sets [PointType](#).
- fs::path [getFilepath](#) ()
Returns path to the file storing the [PointCloud](#).
- void [setFilepath](#) (fs::path filepath)
Sets path to the file storing the [PointCloud](#).
- void [writePointsCount](#) (int n)
Saves number of points to the file storing the [PointCloud](#).
- int [getNumberOfPoints](#) ()
Returns number of points in the [PointCloud](#).
- template<typename T >
void [writeXYZPoint](#) (std::fstream &output, T &point)
Saves the point to the file.
- void [writeXYZNormalPoint](#) (std::fstream &output, [PointXYZ](#) &point)
Saves the point to the file.
- void [writeXYZNormalPoint](#) (std::fstream &output, [PointXYZNormal](#) &point)
Saves the point to the file.
- template<typename T >
void [setPoints](#) (std::vector< T > &points)
Sets points of the [PointCloud](#).
- void [readXYZPoint](#) (std::ifstream &infile, std::vector< [PointXYZ](#) > &points)
Uploads points from the given file.
- void [readXYZPoint](#) (std::ifstream &infile, std::vector< [PointXYZNormal](#) > &points)
Uploads points from the given file.
- void [readXYZNormalPoint](#) (std::ifstream &infile, std::vector< [PointXYZNormal](#) > &points)
Uploads points from the given file.
- void [readXYZNormalPoint](#) (std::ifstream &infile, std::vector< [PointXYZ](#) > &points)
Uploads points from the given file.
- template<typename T >
void [getPoints](#) (std::vector< T > &points)
Returns points stored in the [PointCloud](#).

5.11.1 Constructor & Destructor Documentation

5.11.1.1 `PointCloud::PointCloud (std::string name) [inline]`

Constructs [PointCloud](#) with given name.

Parameters

<i>name</i>	PointCloud name
-------------	---------------------------------

5.11.2 Member Function Documentation

5.11.2.1 `fs::path PointCloud::getFilepath () [inline]`

Returns path to the file storing the [PointCloud](#).

Returns

File path

5.11.2.2 `int PointCloud::getId () const [inline]`

Returns [PointCloud](#) identifier.

Returns

[PointCloud](#) identifier

5.11.2.3 `std::string PointCloud::getName () const [inline]`

Returns [PointCloud](#) name.

Returns

[PointCloud](#) name

5.11.2.4 `int PointCloud::getNumberOfPoints () [inline]`

Returns number of points in the [PointCloud](#).

Returns

Number of points

5.11.2.5 `template<typename T> void PointCloud::getPoints (std::vector< T > & points) [inline]`

Returns points stored in the [PointCloud](#).

Parameters

<i>points</i>	Return parametr storing the points
---------------	------------------------------------

5.11.2.6 `PointType PointCloud::getPointType () const` `[inline]`

Returns PointType.

Returns

PointType (xyz/ xyznormal)

5.11.2.7 `void PointCloud::readXYZNormalPoint (std::ifstream & infile, std::vector< PointXYZNormal > & points)` `[inline]`

Uploads points from the given file.

Parameters

<i>infile</i>	File storing the PointCloud
<i>points</i>	Return parametr storing the uploaded points

5.11.2.8 `void PointCloud::readXYZNormalPoint (std::ifstream & infile, std::vector< PointXYZ > & points)` `[inline]`

Uploads points from the given file.

Parameters

<i>infile</i>	File storing the PointCloud
<i>points</i>	Return parametr storing the uploaded points

5.11.2.9 `void PointCloud::readXYZPoint (std::ifstream & infile, std::vector< PointXYZ > & points)` `[inline]`

Uploads points from the given file.

Parameters

<i>infile</i>	File storing the PointCloud
<i>points</i>	Return parametr storing the uploaded points

5.11.2.10 `void PointCloud::readXYZPoint (std::ifstream & infile, std::vector< PointXYZNormal > & points)`
[inline]

Uploads points from the given file.

Parameters

<i>infile</i>	File storing the PointCloud
<i>points</i>	Return parametr storing the uploaded points

5.11.2.11 `void PointCloud::setFilepath (fs::path filepath)` [inline]

Sets path to the file storing the [PointCloud](#).

Parameters

<i>filepath</i>	File path to be set
-----------------	---------------------

5.11.2.12 `void PointCloud::setId (double id)` [inline]

Sets [PointCloud](#) identifier.

Parameters

<i>id</i>	Identifier to be set
-----------	----------------------

5.11.2.13 `void PointCloud::setName (std::string name)` [inline]

Sets [PointCloud](#) name.

Parameters

<i>name</i>	Name to be set
-------------	----------------

5.11.2.14 `template<typename T> void PointCloud::setPoints (std::vector< T > & points)` [inline]

Sets points of the [PointCloud](#).

Parameters

<i>points</i>	Points to be saved
---------------	--------------------

5.11.2.15 `void PointCloud::setPointType (PointType name)` `[inline]`

Sets PointType.

Parameters

<i>name</i>	PointType to be set (xyz/ xyznormal)
-------------	--------------------------------------

5.11.2.16 `void PointCloud::writePointsCount (int n)` `[inline]`

Saves number of points to the file storing the [PointCloud](#).

Parameters

<i>n</i>	Number of points to be saved
----------	------------------------------

5.11.2.17 `void PointCloud::writeXYZNormalPoint (std::fstream & output, PointXYZ & point)` `[inline]`

Saves the point to the file.

Parameters

<i>output</i>	File where to save the point
<i>point</i>	Point to be saved

5.11.2.18 `void PointCloud::writeXYZNormalPoint (std::fstream & output, PointXYZNormal & point)` `[inline]`

Saves the point to the file.

Parameters

<i>output</i>	File where to save the point
<i>point</i>	Point to be saved

5.11.2.19 `template<typename T> void PointCloud::writeXYZPoint (std::fstream & output, T & point)` `[inline]`

Saves the point to the file.

Parameters

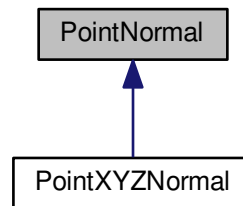
<i>output</i>	File where to save the point
<i>point</i>	Point to be saved

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

- /home/edita/spcl2/src/PointCloud.h

5.12 PointNormal Class Reference

Inheritance diagram for PointNormal:



Public Member Functions

- [PointNormal](#) ()
Constructs [PointNormal](#) with components {0,0,0}.
- [PointNormal](#) (const [PointNormal](#) &p2)
Constructs a copy of the given [PointNormal](#).
- [PointNormal](#) (double x, double y, double z)
Constructs [PointNormal](#) with given components.
- double * [getNormal](#) ()
Returns [PointNormal](#).
- virtual double [getNx](#) ()
Returns [PointNormal](#) component in x-direction.
- virtual double [getNy](#) ()
Returns [PointNormal](#) component in y-direction.
- virtual double [getNz](#) ()
Returns [PointNormal](#) component in z-direction.
- virtual void [setNx](#) (double x)
Sets [PointNormal](#) component in x-direction.
- virtual void [setNy](#) (double y)
Sets [PointNormal](#) component in y-direction.
- virtual void [setNz](#) (double z)
Sets [PointNormal](#) component in z-direction.

Protected Attributes

- double **normal** [3]

5.12.1 Constructor & Destructor Documentation

5.12.1.1 PointNormal::PointNormal (const PointNormal & p2) [inline]

Constructs a copy of the given [PointNormal](#).

Parameters

<i>p2</i>	PointNormal to be copied
-----------	--

5.12.1.2 `PointNormal::PointNormal (double x, double y, double z)` `[inline]`

Constructs [PointNormal](#) with given components.

Parameters

<i>x</i>	Component in x-direction
<i>y</i>	Component in y-direction
<i>z</i>	Component in z-direction

5.12.2 Member Function Documentation

5.12.2.1 `double* PointNormal::getNormal ()` `[inline]`

Returns [PointNormal](#).

Returns

[PointNormal](#)

5.12.2.2 `virtual double PointNormal::getNx ()` `[inline],[virtual]`

Returns [PointNormal](#) component in x-direction.

Returns

[PointNormal](#) component in x-direction

5.12.2.3 `virtual double PointNormal::getNy ()` `[inline],[virtual]`

Returns [PointNormal](#) component in y-direction.

Returns

[PointNormal](#) component in y-direction

5.12.2.4 `virtual double PointNormal::getNz ()` `[inline],[virtual]`

Returns [PointNormal](#) component in z-direction.

Returns

[PointNormal](#) component in z-direction

5.12.2.5 `virtual void PointNormal::setNx (double x)` `[inline],[virtual]`

Sets [PointNormal](#) component in x-direction.

Parameters

x	Component in x-direction
---	--------------------------

5.12.2.6 `virtual void PointNormal::setNy (double y) [inline],[virtual]`

Sets [PointNormal](#) component in y-direction.

Parameters

y	Component in y-direction
---	--------------------------

5.12.2.7 `virtual void PointNormal::setNz (double z) [inline],[virtual]`

Sets [PointNormal](#) component in z-direction.

Parameters

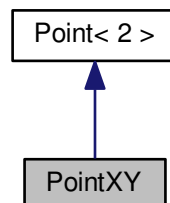
z	Component in z-direction
---	--------------------------

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

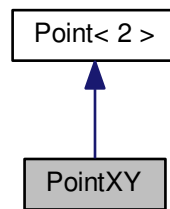
- `/home/edita/spcl2/src/Point.h`

5.13 PointXY Class Reference

Inheritance diagram for PointXY:



Collaboration diagram for PointXY:



Public Member Functions

- [PointXY](#) ()
Constructs [PointXY](#) with coordinates {0,0}.
- [PointXY](#) (double x, double y)
Constructs [PointXY](#) with given coordinates.
- double [getX](#) ()
Returns x coordinate.
- double [getY](#) ()
Returns y coordinate.
- void [setX](#) (double x)
Sets x coordinate.
- void [setY](#) (double y)
Sets y coordinate.

5.13.1 Constructor & Destructor Documentation

5.13.1.1 `PointXY::PointXY (double x, double y) [inline]`

Constructs [PointXY](#) with given coordinates.

Parameters

<i>x</i>	Coordinate in x-direction
<i>y</i>	Coordinate in y-direction

5.13.2 Member Function Documentation

5.13.2.1 `double PointXY::getX () [inline]`

Returns x coordinate.

Returns

Coordinate in x-direction

5.13.2.2 double PointXY::getY () [inline]

Returns y coordinate.

Returns

Coordinate in y-direction

5.13.2.3 void PointXY::setX (double x) [inline]

Sets x coordinate.

Parameters

<i>x</i>	Coordinate in x-direction
----------	---------------------------

5.13.2.4 void PointXY::setY (double y) [inline]

Sets y coordinate.

Parameters

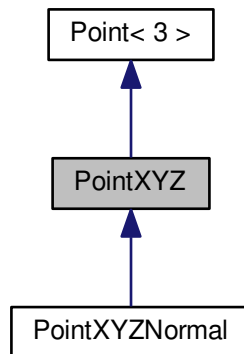
<i>y</i>	Coordinate in y-direction
----------	---------------------------

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

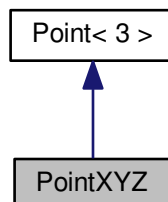
- /home/edita/spcl2/src/Point.h

5.14 PointXYZ Class Reference

Inheritance diagram for PointXYZ:



Collaboration diagram for PointXYZ:



Public Member Functions

- [PointXYZ \(\)](#)
Constructs [PointXYZ](#) with coordinates {0,0,0}.
- [PointXYZ \(const \[PointXYZ\]\(#\) &p2\)](#)
Constructs a copy of the given [PointXYZ](#).
- [PointXYZ \(double x, double y, double z\)](#)
Constructs [PointXYZ](#) with given coordinates.
- virtual void [setCoords](#) (double x, double y, double z)
Sets x, y, z coordinates.
- virtual double [getX](#) ()
Returns x coordinate.
- virtual double [getY](#) ()

- Returns y coordinate.*
- virtual double `getZ` ()
Returns z coordinate.
- virtual void `setX` (double x)
Sets x coordinate.
- virtual void `setY` (double y)
Sets y coordinate.
- virtual void `setZ` (double z)
Sets z coordinate.

5.14.1 Constructor & Destructor Documentation

5.14.1.1 PointXYZ::PointXYZ (const PointXYZ & p2) [inline]

Constructs a copy of the given [PointXYZ](#).

Parameters

<i>p2</i>	Point to be copied
-----------	------------------------------------

5.14.1.2 PointXYZ::PointXYZ (double x, double y, double z) [inline]

Constructs [PointXYZ](#) with given coordinates.

Parameters

<i>x</i>	Coordinate in x-direction
<i>y</i>	Coordinate in y-direction
<i>z</i>	Coordinate in z-direction

5.14.2 Member Function Documentation

5.14.2.1 virtual double PointXYZ::getX () [inline],[virtual]

Returns x coordinate.

Returns

Coordinate in x-direction

5.14.2.2 virtual double PointXYZ::getY () [inline],[virtual]

Returns y coordinate.

Returns

Coordinate in y-direction

5.14.2.3 virtual double PointXYZ::getZ () [inline],[virtual]

Returns z coordinate.

Returns

Coordinate in z-direction

5.14.2.4 virtual void PointXYZ::setCoords (double x, double y, double z) [inline],[virtual]

Sets x, y, z coordinates.

Parameters

<i>x</i>	Coordinate in x-direction
<i>y</i>	Coordinate in y-direction
<i>z</i>	Coordinate in z-direction

5.14.2.5 virtual void PointXYZ::setX (double x) [inline],[virtual]

Sets x coordinate.

Parameters

<i>x</i>	Coordinate in x-direction
----------	---------------------------

5.14.2.6 virtual void PointXYZ::setY (double y) [inline],[virtual]

Sets y coordinate.

Parameters

<i>y</i>	Coordinate in y-direction
----------	---------------------------

5.14.2.7 virtual void PointXYZ::setZ (double z) [inline],[virtual]

Sets z coordinate.

Parameters

<i>z</i>	Coordinate in z-direction
----------	---------------------------

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

- /home/edita/spcl2/src/Point.h

5.15 PointXYZInfo Struct Reference

Public Attributes

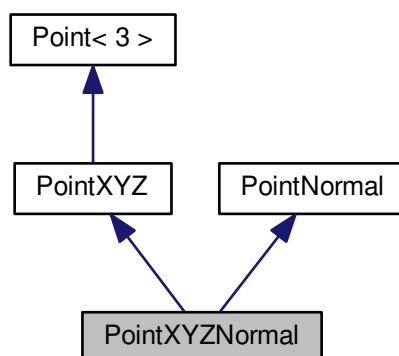
- double `x` = 0.0
X coordinate.
- double `y` = 0.0
Y coordinate.
- double `z` = 0.0
Z coordinate.

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

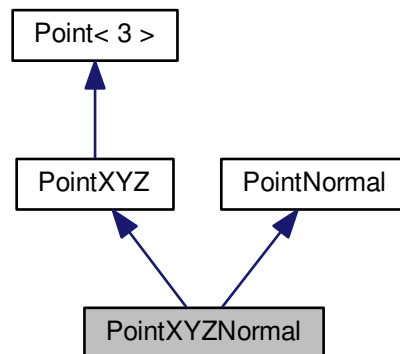
- `/home/edita/spcl2/src/Point.h`

5.16 PointXYZNormal Class Reference

Inheritance diagram for PointXYZNormal:



Collaboration diagram for PointXYZNormal:



Public Member Functions

- [PointXYZNormal](#) ()
Constructs [PointXYZNormal](#) with coordinates {0,0,0}.
- [PointXYZNormal](#) (double x, double y, double z, double nx, double ny, double nz)
Constructs [PointXYZNormal](#) with given coordinates and normal.
- [PointXYZNormal](#) (const [PointXYZNormal](#) &p2)
Constructs a copy of the given [PointXYZNormal](#).

Additional Inherited Members

5.16.1 Constructor & Destructor Documentation

5.16.1.1 [PointXYZNormal::PointXYZNormal](#) (double x, double y, double z, double nx, double ny, double nz) `[inline]`

Constructs [PointXYZNormal](#) with given coordinates and normal.

Parameters

<i>x</i>	Coordinate in x-direction
<i>y</i>	Coordinate in y-direction
<i>z</i>	Coordinate in z-direction
<i>nx</i>	Normal component in x-direction
<i>ny</i>	Normal component in y-direction
<i>nz</i>	Normal component in z-direction

5.16.1.2 PointXYZNormal::PointXYZNormal (const PointXYZNormal & p2) [inline]

Constructs a copy of the given [PointXYZNormal](#).

Parameters

<i>p2</i>	Point to be copied
-----------	------------------------------------

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

- /home/edita/spcl2/src/Point.h

5.17 PointXYZNormalInfo Struct Reference

Public Attributes

- double *x* = 0.0
X coordinate.
- double *y* = 0.0
Y coordinate.
- double *z* = 0.0
Z coordinate.
- double *nx* = 0.0
Normal component in x-direction.
- double *ny* = 0.0
Normal component in y-direction.
- double *nz* = 0.0
Normal component in z-direction.

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

- /home/edita/spcl2/src/Point.h

5.18 Polygon Class Reference

Public Member Functions

- [Polygon](#) ()
Creates empty Polygon.
- void [addPoint](#) ([PointXY](#) &point)
Add single point to Polygon.
- bool [isPointInside](#) ([PointXYZ](#) &point)
Check if given point lies inside of Polygon.
- bool [isPointInside](#) ([PointXY](#) &point)
Check if given point lies inside of Polygon.

5.18.1 Member Function Documentation

5.18.1.1 void Polygon::addPoint (PointXY & point) [inline]

Add single point to [Polygon](#).

Parameters

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

5.18.1.2 `bool Polygon::isPointInside (PointXYZ & point)` `[inline]`

Check if given point lies inside of [Polygon](#).

Parameters

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

Returns

True if point lies inside [Polygon](#)
 False if point lies outside [Polygon](#)

5.18.1.3 `bool Polygon::isPointInside (PointXY & point)` `[inline]`

Check if given point lies inside of [Polygon](#).

Parameters

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

Returns

True if point lies inside [Polygon](#)
 False if point lies outside [Polygon](#)

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

- `/home/edita/spcl2/src/Polygon.h`

5.19 Project Class Reference

Public Member Functions

- [Project](#) (const fs::path &directory)
Creates empty [Project](#) in a given directory path.
- [Project](#) (std::string project_name, const fs::path &full)
Creates empty [Project](#) of a given name in a given directory path.
- std::string & [getFolder](#) ()
Returns [Project](#) folder path.

- double * [getTranslation](#) ()
Returns [Project](#) x, y, z translation.
- void [setTranslation](#) (double tx, double ty, double tz)
Set [Project](#) x, y, z translation.
- void [createProjectFile](#) (const fs::path &path)
Creates [Project](#) JSON file.
- void [save](#) ()
Saves the [Project](#) JSON file.
- void [openProjectFile](#) (const fs::path &path)
Opens the [Project](#) JSON file and read its contents.
- void [addPointCloud](#) ([PointCloud](#) &pc, bool force_save=true)
Add [PointCloud](#) to [Project](#).
- void [update](#) ([PointCloud](#) &p)
Updates [PointCloud](#) after modifying.
- std::vector< [PointCloud](#) > & [getPointClouds](#) ()
Returns all [PointClouds](#).
- [PointCloud](#) & [getPointCloud](#) (int id)
Returns a [PointCloud](#) by identifier.
- void [addRoom](#) ([Room](#) &r, bool force_save=true)
Add [Room](#) to [Project](#).
- void [update](#) ([Room](#) &r)
Updates [Room](#) after modifying.
- std::vector< [Room](#) > & [getRooms](#) ()
Returns all [Rooms](#).
- [Room](#) & [getRoom](#) (int id)
Returns a [Room](#) by identifier.
- void [addFloor](#) ([Floor](#) &f, bool force_save=true)
Add [Floor](#) to [Project](#).
- void [update](#) ([Floor](#) &f)
Updates [Floor](#) after modifying.
- std::vector< [Floor](#) > & [getFloors](#) ()
Returns all [Floors](#).
- [Floor](#) & [getFloor](#) (int id)
Returns a [Floor](#) by identifier.
- void [addPlane](#) ([Plane](#) &p, bool force_save=true)
Add [Plane](#) to [Project](#).
- void [update](#) ([Plane](#) &p)
Updates [Plane](#) after modifying.
- std::vector< [Plane](#) > & [getPlanes](#) ()
Returns all [Planes](#).
- [Plane](#) & [getPlane](#) (int id)
Returns a [Plane](#) by identifier.
- void [addWall](#) ([Wall](#) &p, bool force_save=true)
Add [Wall](#) to [Project](#).
- void [update](#) ([Wall](#) &p)
Updates [Wall](#) after modifying.
- std::vector< [Wall](#) > & [getWalls](#) ()
Returns all [Walls](#).
- [Wall](#) & [getWall](#) (int id)
Returns a [Wall](#) by identifier.
- void [addOpening](#) ([Opening](#) &p, bool force_save=true)

- *Add [Opening](#) to [Project](#).*
- void [update](#) ([Opening](#) &p)
Updates [Opening](#) after modifying.
- std::vector< [Opening](#) > & [getOpenings](#) ()
Returns all openings.
- [Opening](#) & [getOpening](#) (int id)
Returns an [Opening](#) by identifier.
- void [import](#) (std::string filename, bool translate=false)
Imports points from a .pts file (space separated x, y, z(, r, g, b, i) points).

5.19.1 Constructor & Destructor Documentation

5.19.1.1 [Project::Project](#) (const fs::path & *directory*) [inline]

Creates empty [Project](#) in a given directory path.

Parameters

<i>directory</i>	Directory path
------------------	----------------

Returns

5.19.1.2 [Project::Project](#) (std::string *project_name*, const fs::path & *full*) [inline]

Creates empty [Project](#) of a given name in a given directory path.

Parameters

<i>project_name</i>	Project name
<i>full</i>	Directory path

Returns

5.19.2 Member Function Documentation

5.19.2.1 void [Project::addFloor](#) ([Floor](#) & *f*, bool *force_save* = true) [inline]

Add [Floor](#) to [Project](#).

Parameters

<i>f</i>	Floor to add
<i>force_save</i>	Whether to save JSON after adding

5.19.2.2 `void Project::addOpening (Opening & p, bool force_save = true) [inline]`

Add [Opening](#) to [Project](#).

Parameters

<i>p</i>	Opening to add
<i>force_save</i>	Whether to save JSON after adding

5.19.2.3 `void Project::addPlane (Plane & p, bool force_save = true) [inline]`

Add [Plane](#) to [Project](#).

Parameters

<i>p</i>	Plane to add
<i>force_save</i>	Whether to save JSON after adding

5.19.2.4 `void Project::addPointCloud (PointCloud & pc, bool force_save = true) [inline]`

Add [PointCloud](#) to [Project](#).

Parameters

<i>pc</i>	Point cloud to add
<i>force_save</i>	Whether to save JSON after adding

5.19.2.5 `void Project::addRoom (Room & r, bool force_save = true) [inline]`

Add [Room](#) to [Project](#).

Parameters

<i>r</i>	Room to add
<i>force_save</i>	Whether to save JSON after adding

5.19.2.6 `void Project::addWall (Wall & p, bool force_save = true) [inline]`

Add [Wall](#) to [Project](#).

Parameters

<i>p</i>	Wall to add
<i>force_save</i>	Whether to save JSON after adding

5.19.2.7 void Project::createProjectFile (const fs::path & *path*) [inline]

Creates [Project](#) JSON file.

Parameters

<i>path</i>	File path
-------------	-----------

5.19.2.8 Floor& Project::getFloor (int *id*) [inline]

Returns a [Floor](#) by identifier.

Parameters

<i>id</i>	Floor identifier
-----------	----------------------------------

Returns

[Floor](#)

5.19.2.9 std::vector<Floor>& Project::getFloors () [inline]

Returns all Floors.

Returns

Floors

5.19.2.10 std::string& Project::getFolder () [inline]

Returns [Project](#) folder path.

Returns

[Project](#) folder path

5.19.2.11 Opening& Project::getOpening (int *id*) [inline]

Returns an [Opening](#) by identifier.

Parameters

<i>id</i>	Opening identifier
-----------	------------------------------------

Returns

[Opening](#)

5.19.2.12 `std::vector<Opening>& Project::getOpenings () [inline]`

Returns all openings.

Returns

Openings

5.19.2.13 `Plane& Project::getPlane (int id) [inline]`

Returns a [Plane](#) by identifier.

Parameters

<i>id</i>	Plane identifier
-----------	----------------------------------

Returns

[Plane](#)

5.19.2.14 `std::vector<Plane>& Project::getPlanes () [inline]`

Returns all Planes.

Returns

Planes

5.19.2.15 `PointCloud& Project::getPointCloud (int id) [inline]`

Returns a [PointCloud](#) by identifier.

Parameters

<i>id</i>	Point cloud identifier
-----------	--

Returns

[Point](#) cloud

5.19.2.16 `std::vector<PointCloud>& Project::getPointClouds ()` [inline]

Returns all PointClouds.

Returns

[Point](#) clouds

5.19.2.17 `Room& Project::getRoom (int id)` [inline]

Returns a [Room](#) by identifier.

Parameters

<i>id</i>	Room identifier
-----------	---------------------------------

Returns

[Room](#)

5.19.2.18 `std::vector<Room>& Project::getRooms ()` [inline]

Returns all Rooms.

Returns

Rooms

5.19.2.19 `double* Project::getTranslation ()` [inline]

Returns [Project](#) x, y, z translation.

Returns

Translation vector

5.19.2.20 `Wall& Project::getWall (int id)` [inline]

Returns a [Wall](#) by identifier.

Parameters

<i>id</i>	Wall identifier
-----------	---------------------------------

Returns

[Wall](#)

5.19.2.21 `std::vector<Wall>& Project::getWalls ()` `[inline]`

Returns all Walls.

Returns

Walls

5.19.2.22 `void Project::import (std::string filename, bool translate = false)` `[inline]`

Imports points from a .pts file (space separated x, y, z(, r, g, b, i) points).

Parameters

<i>filename</i>	Path to import
<i>translate</i>	Specifies if imported points should be translated to bounding box minimum

5.19.2.23 `void Project::openProjectFile (const fs::path & path)` `[inline]`

Opens the [Project](#) JSON file and read its contents.

Parameters

<i>path</i>	
-------------	--

5.19.2.24 `void Project::setTranslation (double tx, double ty, double tz)` `[inline]`

Set [Project](#) x, y, z translation.

Parameters

<i>tx</i>	Translation in the x direction
<i>ty</i>	Translation in the y direction
<i>tz</i>	Translation in the z direction

5.19.2.25 `void Project::update (PointCloud & p)` `[inline]`

Updates [PointCloud](#) after modifying.

Parameters

p	PointCloud to update
-----	----------------------

5.19.2.26 void Project::update (Room & r) [inline]

Updates Room after modifying.

Parameters

r	Room to update
-----	----------------

5.19.2.27 void Project::update (Floor & f) [inline]

Updates Floor after modifying.

Parameters

f	Floor to update
-----	-----------------

5.19.2.28 void Project::update (Plane & p) [inline]

Updates Plane after modifying.

Parameters

p	Plane to update
-----	-----------------

5.19.2.29 void Project::update (Wall & p) [inline]

Updates Wall after modifying.

Parameters

p	Wall to update
-----	----------------

5.19.2.30 void Project::update (Opening & p) [inline]

Updates Opening after modifying.

Parameters

<i>p</i>	Opening to update
----------	-------------------

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

- /home/edita/spcl2/src/Project.h

5.20 PTXWriter Class Reference

Static Public Member Functions

- static void [write](#) (std::ofstream &file, std::vector< [PointXYZ](#) > &points)
*Writes given points in *.PTX file.*
- static void [write](#) (std::ofstream &file, std::vector< [PointXYZNormal](#) > &points)
*Writes given points in *.PTX file.*

5.20.1 Member Function Documentation

5.20.1.1 `static void PTXWriter::write (std::ofstream & file, std::vector< PointXYZ > & points)` [inline],
[static]

Writes given points in *.PTX file.

Parameters

<i>file</i>	File where to write the output
<i>points</i>	Points to be written

5.20.1.2 `static void PTXWriter::write (std::ofstream & file, std::vector< PointXYZNormal > & points)` [inline],
[static]

Writes given points in *.PTX file.

Parameters

<i>file</i>	File where to write the output
<i>points</i>	Points to be written

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

- /home/edita/spcl2/src/IO/PTXWriter.h

5.21 Room Class Reference

Public Member Functions

- [Room](#) ()
Creates empty [Room](#).
- void [setId](#) (double id)
Sets [Room](#) identifier.
- int [getId](#) () const
Returns room identifier.
- const double * [getMin](#) () const
Returns [Room](#) minimum coordinates.
- void [setMin](#) (double x, double y, double z)
Sets minimum [Room](#) coordinates.
- const double * [getMax](#) () const
Returns [Room](#) maximum coordinates.
- void [setMax](#) (double x, double y, double z)
Sets maximum [Room](#) coordinates.
- void [setFloorLevel](#) (double l)
Sets floor level (height) of the specific [Room](#).
- double [getFloorLevel](#) () const
Returns floor level (height) of the specific [Room](#).
- void [setCeilingLevel](#) (double l)
Sets ceiling level (height) of the specific [Room](#).
- double [getCeilingLevel](#) () const
Returns ceiling level (height) of the specific [Room](#).
- std::vector< int > & [getPointCloudIDs](#) ()
Returns [PointCloud](#) identifiers.
- void [setPointCloudIDs](#) (std::vector< int > pcl_ids)
Sets [PointCloud](#) identifiers.
- const std::vector< [PointCloud](#) > & [getPointClouds](#) () const
Returns all [Room](#) [PointClouds](#).
- std::vector< [PointCloud](#) > & [getPointClouds](#) ()
Returns all [Room](#) [PointClouds](#).
- void [addPointCloud](#) ([PointCloud](#) &pc)
Adds [PointCloud](#) to [Room](#).
- template<typename T >
void [getPoints](#) (std::vector< T > &points)
Returns all [Room](#) points.
- std::vector< int > & [getPlaneIDs](#) ()
Returns [Plane](#) identifiers.
- void [setPlaneIDs](#) (std::vector< int > pcl_ids)
Sets [Plane](#) identifiers.
- std::vector< [Plane](#) > & [getPlanes](#) ()
Returns all [Planes](#) found in a [Room](#).
- const std::vector< [Plane](#) > & [getPlanes](#) () const
Returns all [Planes](#) found in a [Room](#).
- void [addPlane](#) ([Plane](#) &pc)
Adds a [Plane](#) to the [Room](#).
- void [setArea](#) (double area)
Sets floor plan area of the [Room](#).
- double [getArea](#) () const
Returns floor plan area of the [Room](#).

5.21.1 Member Function Documentation

5.21.1.1 void Room::addPlane (Plane & *pc*) [inline]

Adds a [Plane](#) to the [Room](#).

Parameters

<i>pc</i>	Plane to be added
-----------	-----------------------------------

5.21.1.2 void Room::addPointCloud (PointCloud & *pc*) [inline]

Adds [PointCloud](#) to [Room](#).

Parameters

<i>pc</i>	PointCloud to be added
-----------	--

5.21.1.3 double Room::getArea () const [inline]

Returns floor plan area of the [Room](#).

Returns

Area [m^2]

5.21.1.4 double Room::getCeilingLevel () const [inline]

Returns ceiling level (height) of the specific [Room](#).

Returns

Ceiling level

5.21.1.5 double Room::getFloorLevel () const [inline]

Returns floor level (height) of the specific [Room](#).

Returns

[Floor](#) level

5.21.1.6 `int Room::getId () const [inline]`

Returns room identifier.

Returns

[Room](#) identifier

5.21.1.7 `const double* Room::getMax () const [inline]`

Returns [Room](#) maximum coordinates.

Returns

Maximum [Room](#) coordinates

5.21.1.8 `const double* Room::getMin () const [inline]`

Returns [Room](#) minimum coordinates.

Returns

Minimum [Room](#) coordinates

5.21.1.9 `std::vector<int>& Room::getPlaneIDs () [inline]`

Returns [Plane](#) identifiers.

Returns

[Plane](#) identifiers

5.21.1.10 `std::vector<Plane>& Room::getPlanes () [inline]`

Returns all [Planes](#) found in a [Room](#).

Returns

[Room](#) [Planes](#)

5.21.1.11 `const std::vector<Plane>& Room::getPlanes () const [inline]`

Returns all [Planes](#) found in a [Room](#).

Returns

[Room](#) [Planes](#)

5.21.1.12 `std::vector<int>& Room::getPointCloudIDs () [inline]`

Returns [PointCloud](#) identifiers.

Returns

[PointCloud](#) identifiers

5.21.1.13 `const std::vector<PointCloud>& Room::getPointClouds () const [inline]`

Returns all [Room](#) PointClouds.

Returns

[Room](#) PointClouds

5.21.1.14 `std::vector<PointCloud>& Room::getPointClouds () [inline]`

Returns all [Room](#) PointClouds.

Returns

[Room](#) PointClouds

5.21.1.15 `template<typename T> void Room::getPoints (std::vector< T > & points) [inline]`

Returns all [Room](#) points.

Template Parameters

<i>T</i>	Point type
----------	----------------------------

Parameters

<i>points</i>	Room Points
---------------	-----------------------------

5.21.1.16 `void Room::setArea (double area) [inline]`

Sets floor plan area of the [Room](#).

Parameters

<i>Area</i>	[m^2]
-------------	-----------

5.21.1.17 `void Room::setCeilingLevel (double l) [inline]`

Sets ceiling level (height) of the specific [Room](#).

Parameters

<i>l</i>	Floor level
----------	-------------

5.21.1.18 `void Room::setFloorLevel (double l) [inline]`

Sets floor level (height) of the specific [Room](#).

Parameters

<i>l</i>	Floor level
----------	-------------

5.21.1.19 `void Room::setId (double id) [inline]`

Sets [Room](#) identifier.

Parameters

<i>id</i>	Room identifier
-----------	-----------------

5.21.1.20 `void Room::setMax (double x, double y, double z) [inline]`

Sets maximum [Room](#) coordinates.

Parameters

<i>x</i>	Maximum x coordinate
<i>y</i>	Maximum y coordinate
<i>z</i>	Maximum z coordinate

5.21.1.21 `void Room::setMin (double x, double y, double z) [inline]`

Sets minimum [Room](#) coordinates.

Parameters

<i>x</i>	Minimal x coordinate
<i>y</i>	Minimal y coordinate
<i>z</i>	Minimal z coordinate

5.21.1.22 void Room::setPlaneIds (std::vector< int > *pcl_ids*) [inline]

Sets [Plane](#) identifiers.

Parameters

<i>pcl_ids</i>	Plane identifiers
----------------	-----------------------------------

5.21.1.23 void Room::setPointCloudIds (std::vector< int > *pcl_ids*) [inline]

Sets [PointCloud](#) identifiers.

Parameters

<i>pcl_ids</i>	PointCloud identifiers
----------------	--

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

- /home/edita/spcl2/src/Room.h

5.22 Vector3 Class Reference

Public Member Functions

- [Vector3](#) ()
Creates a zero vector.
- [Vector3](#) (double vx, double vy, double vz)
Creates a new vector with specified coordinates.
- [Vector3](#) (double vx, double vy)
Creates a new 2D vector with specified coordinates.
- [Vector3](#) ([PointXYZ](#) &p)
Creates vector from 3D point.
- [Vector3](#) ([PointXYZNormal](#) &p)
Creates vector from 3D point with normals.
- [Vector3](#) operator+ (const [Vector3](#) &o)
Operator for vector addition.
- [Vector3](#) & operator+= (const [Vector3](#) &o)
Operator for addition to a vector.
- [Vector3](#) operator- ()
Operator for vector inversion.
- [Vector3](#) operator- (const [Vector3](#) o)
Operator for vector subtraction.
- [Vector3](#) & operator-= (const [Vector3](#) o)
Operator for subtraction from a vector.
- [Vector3](#) operator* (const double s)
Operator for vector multiplication.
- [Vector3](#) & operator*= (const double s)

- Operator for vector multiplication.*

 - `Vector3 operator/` (const double s)
Operator for vector division.
 - `Vector3 & operator/=` (const double s)
Operator for vector division.
 - double `length` ()
Calculates length of a vector.
 - double `length_sqr` ()
Calculates squared length of a vector.
 - `Vector3 normalized` ()
Normalize vector.
 - `Vector3 & normalize` ()
Normalize vector.
 - double `operator*` (const `Vector3` o)
Operator for dot product of two vectors.
 - `Vector3 operator^` (const `Vector3` o)
Calculates cross product of two vectors.
 - `Vector3 & operator^=` (const `Vector3` o)
Calculates cross product of two vectors.

Public Attributes

- double `x`
X component.
- double `y`
Y component.
- double `z`
Z component.

5.22.1 Constructor & Destructor Documentation

5.22.1.1 `Vector3::Vector3 (double vx, double vy, double vz)` [`inline`]

Creates a new vector with specified coordinates.

Parameters

<code>vx</code>	X component
<code>vy</code>	Y component
<code>vz</code>	Z component

5.22.1.2 `Vector3::Vector3 (double vx, double vy)` [`inline`]

Creates a new 2D vector with specified coordinates.

Parameters

<code>vx</code>	X component
<code>vy</code>	Y component

5.22.1.3 `Vector3::Vector3 (PointXYZ & p)` `[inline]`

Creates vector from 3D point.

Parameters

<code>p</code>	PointXYZ
----------------	--------------------------

5.22.1.4 `Vector3::Vector3 (PointXYZNormal & p)` `[inline]`

Creates vector from 3D point with normals.

Parameters

<code>p</code>	PointXYZNormal
----------------	--------------------------------

5.22.2 Member Function Documentation**5.22.2.1** `double Vector3::length ()` `[inline]`

Calculates length of a vector.

Returns

Vector length

5.22.2.2 `double Vector3::length_sqr ()` `[inline]`

Calculates squared length of a vector.

Returns

Squared vector length

5.22.2.3 `Vector3& Vector3::normalize ()` `[inline]`

Normalize vector.

Returns

Normalized vector

5.22.2.4 Vector3 Vector3::normalized () [inline]

Normalize vector.

Returns

Normalized vector

5.22.2.5 Vector3 Vector3::operator*(const double s) [inline]

Operator for vector multiplication.

Parameters

s	Scalar
---	--------

Returns

New vector

5.22.2.6 double Vector3::operator*(const Vector3 o) [inline]

Operator for dot product of two vectors.

Parameters

o	Vector
---	--------

Returns

Dot product

5.22.2.7 Vector3& Vector3::operator*=(const double s) [inline]

Operator for vector multiplication.

Parameters

s	Scalar
---	--------

Returns

Current vector

5.22.2.8 Vector3 Vector3::operator+ (const Vector3 & o) [inline]

Operator for vector addition.

Parameters

<i>o</i>	Vector to be added
----------	--------------------

Returns

New Vector

5.22.2.9 Vector3& Vector3::operator+= (const Vector3 & o) [inline]

Operator for addition to a vector.

Parameters

<i>o</i>	Vector to be added
----------	--------------------

Returns

Current vector

5.22.2.10 Vector3 Vector3::operator- () [inline]

Operator for vector inversion.

Returns

New vector with reversed signs

5.22.2.11 Vector3 Vector3::operator- (const Vector3 o) [inline]

Operator for vector subtraction.

Parameters

<i>o</i>	Vector to be subtracted
----------	-------------------------

Returns

New vector

5.22.2.12 `Vector3& Vector3::operator-= (const Vector3 o) [inline]`

Operator for subtraction from a vector.

Parameters

<i>o</i>	Vector to subtract
----------	--------------------

Returns

Current vector

5.22.2.13 `Vector3 Vector3::operator/ (const double s) [inline]`

Operator for vector division.

Parameters

<i>s</i>	Scalar
----------	--------

Returns

New vector

5.22.2.14 `Vector3& Vector3::operator/= (const double s) [inline]`

Operator for vector division.

Parameters

<i>s</i>	Scalar
----------	--------

Returns

Current vector

5.22.2.15 `Vector3 Vector3::operator^ (const Vector3 o) [inline]`

Calculates cross product of two vectors.

Parameters

<i>o</i>	Vector
----------	--------

Returns

Cross product

5.22.2.16 Vector3& Vector3::operator^= (const Vector3 o) [inline]

Calculates cross product of two vectors.

Parameters

<i>o</i>	Vector
----------	--------

Returns

Current vector

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

- /home/edita/spcl2/src/Vector3.h

5.23 Wall Class Reference

Public Member Functions

- [Wall](#) ()
Creates empty Wall.
- int [getId](#) () const
Returns Wall identifier.
- void [setId](#) (int id)
Sets Wall identifier.
- int [getFloorId](#) () const
Returns Floor identifier.
- void [setFloorId](#) (int id)
Sets Floor identifier.
- bool [isOuter](#) () const
Checks if wall is marked as outer.
- void [setOuter](#) (bool outer)
Changes outer flag.
- std::vector< int > & [getPlaneIDs](#) ()
Returns Plane identifiers.
- void [setPlaneIDs](#) (std::vector< int > pcl_ids)
Sets Plane identifiers.
- const std::vector< [Plane](#) > & [getPlanes](#) () const
Returns all Planes.
- std::vector< [Plane](#) > & [getPlanes](#) ()
Returns all Planes.
- void [addPlane](#) ([Plane](#) &pc)

- Adds a [Plane](#) to the [Wall](#).*

 - `std::vector< int > & getRoomIDs ()`
Returns [Room](#) identifiers.
 - `void setRoomIDs (std::vector< int > pcl_ids)`
Sets [Room](#) identifiers.
 - `const std::vector< Room > & getRooms () const`
Returns all [Rooms](#).
 - `std::vector< Room > & getRooms ()`
Returns all [Rooms](#).
 - `void addRoom (Room &room)`
Adds [Room](#) to the [Wall](#).
 - `template<typename T >`
`void getPoints (std::vector< T > &points)`
Returns all points of the [Wall](#). Iterates over all [Planes](#) and pushes its [Points](#) into a vector.

5.23.1 Member Function Documentation

5.23.1.1 void Wall::addPlane ([Plane](#) & *pc*) [*inline*]

Adds a [Plane](#) to the [Wall](#).

Parameters

<i>pc</i>	Plane
-----------	-----------------------

5.23.1.2 void Wall::addRoom ([Room](#) & *room*) [*inline*]

Adds [Room](#) to the [Wall](#).

Parameters

<i>room</i>	Room
-------------	----------------------

5.23.1.3 int Wall::getFloorId () const [*inline*]

Returns [Floor](#) identifier.

Returns

[Floor](#) identifier

5.23.1.4 int Wall::getId () const [*inline*]

Returns [Wall](#) identifier.

Returns

[Wall](#) identifier

5.23.1.5 `std::vector<int>& Wall::getPlaneIDs () [inline]`

Returns [Plane](#) identifiers.

Returns

[Plane](#) identifiers

5.23.1.6 `const std::vector<Plane>& Wall::getPlanes () const [inline]`

Returns all Planes.

Returns

Planes

5.23.1.7 `std::vector<Plane>& Wall::getPlanes () [inline]`

Returns all Planes.

Returns

Planes

5.23.1.8 `template<typename T> void Wall::getPoints (std::vector< T > & points) [inline]`

Returns all points of the [Wall](#). Iterates over all Planes and pushes its Points into a vector.

Template Parameters

<i>T</i>	Point type
----------	----------------------------

Parameters

<i>points</i>	Points
---------------	--------

5.23.1.9 `std::vector<int>& Wall::getRoomIDs () [inline]`

Returns [Room](#) identifiers.

Returns

[Room](#) identifiers

5.23.1.10 `const std::vector<Room>& Wall::getRooms () const [inline]`

Returns all Rooms.

Returns

Rooms

5.23.1.11 `std::vector<Room>& Wall::getRooms () [inline]`

Returns all Rooms.

Returns

Rooms

5.23.1.12 `bool Wall::isOuter () const [inline]`

Checks if wall is marked as outer.

Returns

True if wall is marked as outer, it probably has just a single plane
False if wall is marked as inner, it probably has 2 planes (or more)

5.23.1.13 `void Wall::setFloorId (int id) [inline]`

Sets [Floor](#) identifier.

Parameters

<i>id</i>	Floor identifier
-----------	----------------------------------

5.23.1.14 `void Wall::setId (int id) [inline]`

Sets [Wall](#) identifier.

Parameters

<i>id</i>	Wall identifier
-----------	---------------------------------

5.23.1.15 `void Wall::setOuter (bool outer) [inline]`

Changes outer flag.

Parameters

<i>outer</i>	Is wall outer?
--------------	----------------

5.23.1.16 void Wall::setPlaneIDs (std::vector< int > *pcl_ids*) [inline]

Sets [Plane](#) identifiers.

Parameters

<i>pcl_ids</i>	Plane identifiers
----------------	-----------------------------------

5.23.1.17 void Wall::setRoomIDs (std::vector< int > *pcl_ids*) [inline]

Sets [Room](#) identifiers.

Parameters

<i>pcl_ids</i>	Room identifiers
----------------	----------------------------------

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

- /home/edita/spcl2/src/Wall.h

Chapter 6

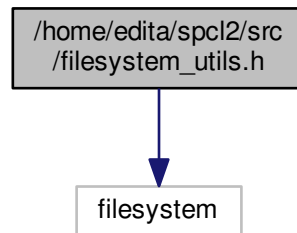
File Documentation

6.1 /home/edita/spcl2/src/filesystem_utils.h File Reference

Set of functions supporting filesystem operations.

```
#include <filesystem>
```

Include dependency graph for filesystem_utils.h:



Functions

- `bool directory_exists (const fs::path &p, fs::file_status s=fs::file_status{})`
Check if a given directory path exists.

6.1.1 Detailed Description

Set of functions supporting filesystem operations.

6.1.2 Function Documentation

6.1.2.1 `bool directory_exists (const fs::path & p, fs::file_status s = fs::file_status{})`

Check if a given directory path exists.

Parameters

<i>p</i>	Path to directory
<i>s</i>	File status

Returns

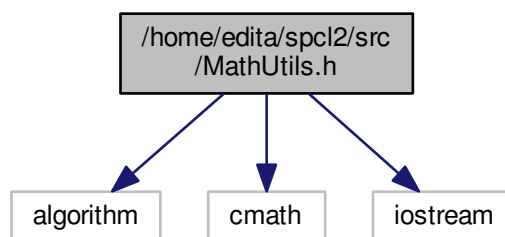
True if directory exists

False if directory does not exist

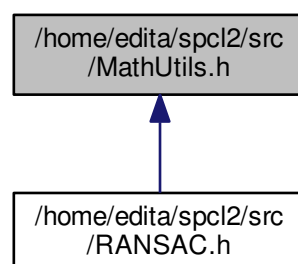
6.2 /home/edita/spcl2/src/MathUtils.h File Reference

Set of functions supporting mathematic operations.

```
#include <algorithm>
#include <cmath>
#include <iostream>
#include "BoundingBox.h"
Include dependency graph for MathUtils.h:
```



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



Functions

- double [det2](#) (double a1, double b1, double c1, double d1)
Calculates the determinant of the given 2x2 matrix.
- double [det3](#) (double matrix[3][3])
Calculates the determinant of the given 3x3 matrix.
- void [eig3](#) (double matrix[3][3], double lambda, double(&result)[3])
Calculates eigenvector of the given 3x3 matrix for the given eigenvalue.
- void [cubicEq](#) (double au, double bu, double cu, double du, double(&result)[3])
Calculates the solution of a cubic equation ($ax^3 + bx^2 + cx + d = 0$).
- template<typename T >
void [averageD](#) (std::vector< T > &points, double(&v1)[3], double(&averageD))
Calculates the average coefficient d for the plane ($ax + bx + cx + d = 0$) given by normal vector and interpolated through the given points.
- template<typename T >
void [planeFromPoints](#) (std::vector< T > &points, std::vector< int > &indexes, double(&v1)[3], double &averageD)
Interpolates the plane through the given points.
- template<typename T >
void [planeFromPoints](#) (std::vector< T > &points, double(&v1)[3], double(&v2)[3], double(&v3)[3], double &averageD)
Interpolates the plane through the given points.
- double [angleBetweenPlanes](#) (double a1, double b1, double c1, double a2, double b2, double c2)
Calculates the angle between two planes ($ax + bx + cx + d = 0$).
- double [angleBetweenPlanes](#) (double n1[3], double n2[3])
Calculates the angle between two planes given by normal vectors.
- double [dot](#) (double ax, double ay, double az, double bx, double by, double bz)
Calculates the dot product of two vectors.
- void [cross](#) (double(&a)[3], double(&b)[3], double(&result)[3])
Calculates the cross product of two vectors.
- void [normalize](#) (double(&vec)[3])
Normalizes the given vector.
- template<typename T >
void [rotatePointsToLocal](#) (std::vector< T > &points, double(&locx)[3], double(&locy)[3])
Rotates points in the local coordinate system ($z = 0$).
- template<typename T >
void [rotatePoint](#) (T &pt, double(&v1)[3], double(&v2)[3], double(&v3)[3])
Rotates point in the local coordinate system given by its base vectors.
- template<typename T >
void [rotatePoints](#) (std::vector< T > &points, double(&v1)[3], double(&v2)[3], double(&v3)[3])
Rotates points in the local coordinate system given by its base vectors.
- template<typename T >
void [moveCSorigin](#) (std::vector< T > &points, T O)
Recalculates the coordinates of the points when the new origin of the coordinate system is set.
- template<typename T >
void [moveBackCSorigin](#) (std::vector< T > &points, T O)
Recalculates the coordinates of the points when the origin of the coordinate system is reset from the given O to (0, 0, 0).
- template<typename T >
void [giveLocalCS](#) (std::vector< T > &points, double(&locx)[3], double(&locy)[3], double(&locz)[3])
Calculates local coordinate system of the plane defined by the given points.
- void [giveLtoGVectorsFromLocalCS](#) (double(&locx)[3], double(&locy)[3], double(&locz)[3], double(&l2g1)[3], double(&l2g2)[3], double(&l2g3)[3])

Calculates the global-to-local (G2L) transformation vectors to local-to-global (L2G) transformation vectors.

- `template<typename T >`
`double giveAverageCoord (std::vector< T > &points, int cIdx)`

Calculates the average coordinate of the given points.

- `template<typename T >`
`void projectPointsToXYPlane (std::vector< T > &points)`

Sets the z-coordinate of the given points to the zero.

6.2.1 Detailed Description

Set of functions supporting mathematic operations.

6.2.2 Function Documentation

6.2.2.1 `double angleBetweenPlanes (double a1, double b1, double c1, double a2, double b2, double c2)`

Calculates the angle between two planes ($ax + bx + cx + d = 0$).

Parameters

<i>a1</i>	Coefficient a of the first plane
<i>b1</i>	Coefficient b of the first plane
<i>c1</i>	Coefficient c of the first plane
<i>a1</i>	Coefficient a of the second plane
<i>b1</i>	Coefficient b of the second plane
<i>c1</i>	Coefficient c of the second plane

Returns

Angle between the planes

6.2.2.2 `double angleBetweenPlanes (double n1[3], double n2[3])`

Calculates the angle between two planes given by normal vectors.

Parameters

<i>n1</i>	Normal vector of the first plane
<i>n2</i>	Normal vector of the second plane

Returns

Angle between the planes

6.2.2.3 `template<typename T > void averageD (std::vector< T > & points, double(&) v1[3], double & averageD)`

Calculates the average coefficient d for the plane ($ax + bx + cx + d = 0$) given by normal vector and interpolated through the given points.

Parameters

<i>points</i>	Plane points
<i>v1</i>	Plane normal vector
<i>averageD</i>	Return parameter storing the calculated coefficient

6.2.2.4 `void cross (double(&) a[3], double(&) b[3], double(&) result[3])`

Calculates the cross product of two vectors.

Parameters

<i>a</i>	The first vector
<i>b</i>	The second vector
<i>result</i>	Return parameter storing the resulting cross product

6.2.2.5 `void cubicEq (double au, double bu, double cu, double du, double(&) result[3])`

Calculates the solution of a cubic equation ($ax^3 + bx^2 + cx + d = 0$).

Parameters

<i>au</i>	Coefficient a
<i>bu</i>	Coefficient b
<i>cu</i>	Coefficient c
<i>du</i>	Coefficient d
<i>result</i>	Return parameter storing the calculated solution

6.2.2.6 `double det2 (double a1, double b1, double c1, double d1)`

Calculates the determinant of the given 2x2 matrix.

Parameters

<i>a1</i>	Matrix element (1,1)
<i>b1</i>	Matrix element (1,2)
<i>c1</i>	Matrix element (2,1)
<i>d1</i>	Matrix element (2,2)

Returns

Determinant

6.2.2.7 `double det3 (double matrix[3][3])`

Calculates the determinant of the given 3x3 matrix.

Parameters

<i>matrix</i>	3x3 matrix
---------------	------------

Returns

Determinant

6.2.2.8 `double dot (double ax, double ay, double az, double bx, double by, double bz)`

Calculates the dot product of two vectors.

Parameters

<i>ax</i>	X component of the first vector
<i>ay</i>	Y component of the first vector
<i>az</i>	Z component of the first vector
<i>bx</i>	X component of the second vector
<i>by</i>	Y component of the second vector
<i>bz</i>	Z component of the second vector

Returns

Resulting dot product

6.2.2.9 `void eig3 (double matrix[3][3], double lambda, double(&) result[3])`

Calculates eigenvector of the given 3x3 matrix for the given eigenvalue.

Parameters

<i>matrix</i>	3x3 matrix
<i>lambda</i>	Eigenvalue
<i>result</i>	Return parameter storing the calculated eigenvector

6.2.2.10 `template<typename T> double giveAverageCoord (std::vector< T > & points, int cIndex)`

Calculates the average coordinate of the given points.

Parameters

<i>points</i>	The points for calculation of the average coordinate
<i>index</i>	Index specifying which coordinate to calculate (0~x, 1~y, 2~z)

Returns

The average coordinate calculated

6.2.2.11 `template<typename T> void giveLocalCS (std::vector< T > & points, double(&) locx[3], double(&) locy[3], double(&) locz[3])`

Calculates local coordinate system of the plane defined by the given points.

Parameters

<i>points</i>	Points defining the plane for which to calculate the local coordinate system
<i>locx</i>	Return parameter storing local x-axis (in plane, horizontal)
<i>locy</i>	Return parameter storing local y-axis (in plane, vertical)
<i>locz</i>	Return parameter storing local z-axis (normal vector)

6.2.2.12 `void giveLtoGVectorsFromLocalCS (double(&) locx[3], double(&) locy[3], double(&) locz[3], double(&) l2g1[3], double(&) l2g2[3], double(&) l2g3[3])`

Calculates the global-to-local (G2L) transformation vectors to local-to-global (L2G) transformation vectors.

Parameters

<i>locx</i>	The first G2L transformation vector
<i>locy</i>	The second G2L transformation vector
<i>locz</i>	The third G2L transformation vector
<i>l2g1</i>	In-out parameter storing the first L2G transformation vector $\{locx(1), locy(1), locz(1)\}$
<i>l2g1</i>	In-out parameter storing the second L2G transformation vector $\{locx(2), locy(2), locz(2)\}$
<i>l2g1</i>	In-out parameter storing the third L2G transformation vector $\{locx(3), locy(3), locz(3)\}$

6.2.2.13 `template<typename T> void moveBackCSorigin (std::vector< T > & points, T O)`

Recalculates the coordinates of the points when the origin of the coordinate system is reset from the given O to (0, 0, 0).

Parameters

<i>pt</i>	In-Out parameter storing the points to be transformed
<i>O</i>	Actual origin coordinates

6.2.2.14 `template<typename T> void moveCSorigin (std::vector< T > & points, T O)`

Recalculates the coordinates of the points when the new origin of the coordinate system is set.

Parameters

<i>pt</i>	In-Out parameter storing the points to be transformed
<i>O</i>	New origin coordinates

6.2.2.15 `void normalize (double(&) vec[3])`

Normalizes the given vector.

Parameters

<i>vec</i>	In-Out parameter storing the vector to be normalized
------------	--

6.2.2.16 `template<typename T> void planeFromPoints (std::vector< T > & points, std::vector< int > & indexes, double(&) v1[3], double & averageD)`

Interpolates the plane through the given points.

Parameters

<i>points</i>	Points for the calculation
<i>indexes</i>	Indexes specifying which points to use for the plane calculation
<i>v1</i>	Return parametr storing the normal vector of the plane
<i>averageD</i>	Coefficient d of the plane equation ($ax + bx + cx + d = 0$)

6.2.2.17 `template<typename T> void planeFromPoints (std::vector< T > & points, double(&) v1[3], double(&) v2[3], double(&) v3[3], double & averageD)`

Interpolates the plane through the given points.

Parameters

<i>points</i>	Points through which to interpolate the plane
<i>v1</i>	Return parametr storing the normal vector of the plane
<i>v2</i>	Return parametr storing the first in-plane vector

Parameters

<i>v3</i>	Return parametr storing the second in-plane vector
<i>averageD</i>	Coefficient d of the plane equation ($ax + bx + cx + d = 0$)

6.2.2.18 `template<typename T> void projectPointsToXYPlane (std::vector< T > & points)`

Sets the z-coordinate of the given points to the zero.

Parameters

<i>points</i>	In-Out parameter storing the points to be projected
---------------	---

6.2.2.19 `template<typename T> void rotatePoint (T & pt, double(&) v1[3], double(&) v2[3], double(&) v3[3])`

Rotates point in the local coordinate system given by its base vectors.

Parameters

<i>pt</i>	In-Out parameter storing the point to be rotated
<i>v1</i>	The first base vector of the local coordinate system
<i>v2</i>	The second base vector of the local coordinate system
<i>v3</i>	The third base vector of the local coordinate system

6.2.2.20 `template<typename T> void rotatePoints (std::vector< T > & points, double(&) v1[3], double(&) v2[3], double(&) v3[3])`

Rotates points in the local coordinate system given by its base vectors.

Parameters

<i>pt</i>	In-Out parameter storing the points to be rotated
<i>v1</i>	The first base vector of the local coordinate system
<i>v2</i>	The second base vector of the local coordinate system
<i>v3</i>	The third base vector of the local coordinate system

6.2.2.21 `template<typename T> void rotatePointsToLocal (std::vector< T > & points, double(&) locx[3], double(&) locy[3])`

Rotates points in the local coordinate system (z = 0).

Parameters

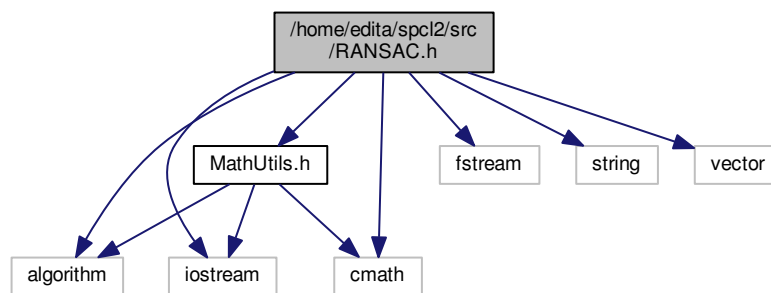
<i>points</i>	In-Out parameter storing the points to be rotated
<i>locx</i>	Local x vector
<i>locy</i>	Local y vector

6.3 /home/edita/spcl2/src/RANSAC.h File Reference

Set of functions supporting RANSAC algorithm.

```
#include <algorithm>
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <cmath>
#include "Point.h"
#include "MathUtils.h"
```

Include dependency graph for RANSAC.h:



Functions

- void [plane_parameters](#) (double(&point_coord)[3][3], double(&plane)[4])
Calculates Plane coefficients from 3 points.
- template<typename T >
void [random_three_pnt](#) (std::vector< T > &data, double(&ThreePoint)[3][3], double thres)
Select 3 random points from the point list which are likely to form vertical plane.
- template<typename T >
void [findPlanes](#) (std::vector< T > &dataset, std::vector< std::vector< T >> &output)
Find vertical planes in a given point cloud.

6.3.1 Detailed Description

Set of functions supporting RANSAC algorithm.

6.3.2 Function Documentation

6.3.2.1 template<typename T > void [findPlanes](#) (std::vector< T > & dataset, std::vector< std::vector< T >> & output)

Find vertical planes in a given point cloud.

Template Parameters

<i>T</i>	Point type
----------	------------

Parameters

<i>dataset</i>	Points for plane finding
<i>output</i>	Vector of vectors containing each plane's points

6.3.2.2 void plane_parameters (double(&) *point_coord*[3][3], double(&) *plane*[4])

Calculates Plane coefficients from 3 points.

Parameters

<i>point_coord</i>	2D Array of 3 points 3D coordinates
<i>plane</i>	Array of 4 plane coefficients

6.3.2.3 template<typename T > void random_three_pnt (std::vector< T > & *data*, double(&) *ThreePoint*[3][3], double *thres*)

Select 3 random points from the point list which are likely to form vertical plane.

Template Parameters

<i>T</i>	Point type
----------	------------

Parameters

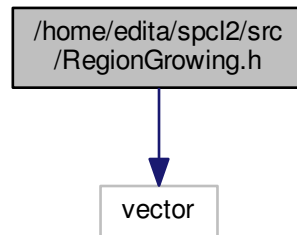
<i>data</i>	Points to select from
<i>ThreePoint</i>	2D Array of 3D Point coordinates
<i>thres</i>	Minimal distance for point selection

6.4 /home/edita/spcl2/src/RegionGrowing.h File Reference

Set of functions for region growing.

```
#include <vector>
#include "BitGrid.h"
```

Include dependency graph for RegionGrowing.h:



Functions

- bool `getTrials` (`BitGrid &g`, `std::vector< int > &trials`, `int index`)
Calculates `BitGrid` cell indexes one cell around the given cell.
- bool `getTrials2` (`BitGrid &g`, `std::vector< int > &trials`, `int index`)
Calculates `BitGrid` cell indexes two cells around the given cell.
- bool `grow` (`BitGrid &g`, `std::vector< int > &room`, `int index`)
Adds the cell index to the vector if the cell is active and deactivates the cell.
- void `regionGrowing` (`BitGrid &g`, `std::vector< BitGrid > &segList`, `double minArea`, `int cellStep=1`)
Search for the active segments with minimal area in the given grid.
- void `regionGrowingAround` (`BitGrid &g`, `std::vector< std::vector< int >> trialsVec`, `std::vector< BitGrid > &segList`)
Search for the region of active cells in the given grid around cells given by the trial vectors.
- bool `inverseGrow` (`BitGrid &g`, `std::vector< int > &segment`, `int index`)
Adds the cell index to the vector if the cell is inactive and activates the cell.
- void `inverseRegionGrowing` (`BitGrid &initialGrid`, `std::vector< BitGrid > &segList`, `std::vector< BitGrid > &boundaries`, `double minArea`)
Search for the inactive segments with minimal area in the given grid.
- `std::vector< bool >` `boundaryGrowing2` (`BitGrid &intermedPts`, `std::vector< BitGrid > &boundaries`, `std::vector< std::vector< int >> &indexes`, `std::vector< double > minIntermedPts`)
Search for the active cells in the grid around (+-1 cell) given boundaries segments with minimal area in the given grid and evaluates if the minimal number of cells is found.
- `std::vector< bool >` `boundaryGrowing` (`BitGrid &intermedPts`, `std::vector< BitGrid > &boundaries`, `std::vector< std::vector< int >> &indexes`, `double minIntermedPts`)

6.4.1 Detailed Description

Set of functions for region growing.

6.4.2 Function Documentation

6.4.2.1 `std::vector<bool> boundaryGrowing2 (BitGrid & intermedPts, std::vector< BitGrid > & boundaries, std::vector< std::vector< int >> & indexes, std::vector< double > minIntermedPts)`

Search for the active cells in the grid around (+-1 cell) given boundaries segments with minimal area in the given grid and evaluates if the minimal number of cells is found.

Parameters

<i>intermedPts</i>	BitGrid in which to search for the active cells
<i>boundaries</i>	Vector of the boundaries around which to search for the active cells
<i>indexes</i>	Return parameter storing the indexes of the bands found
<i>minIntermedPts</i>	Minimal ratio of the boundary cells over the found active cells

Returns

True if sufficient number of active cells is found
 False if insufficient number of active cells is found

6.4.2.2 `bool getTrials (BitGrid & g, std::vector< int > & trials, int index)`

Calculates [BitGrid](#) cell indexes one cell around the given cell.

Parameters

<i>g</i>	BitGrid for which to calculate the indexes
<i>trials</i>	Return parameter storing the calculated indexes
<i>index</i>	Index of the cell around which to search for the neighbours

Returns

True if neighbouring indexes are calculated
 False for invalid initial index

6.4.2.3 `bool getTrials2 (BitGrid & g, std::vector< int > & trials, int index)`

Calculates [BitGrid](#) cell indexes two cells around the given cell.

Parameters

<i>g</i>	BitGrid for which to calculate the indexes
<i>trials</i>	Return parameter storing the calculated indexes
<i>index</i>	Index of the cell around which to search for the neighbours

Returns

True if neighbouring indexes are calculated
 False for invalid initial index

6.4.2.4 `bool grow (BitGrid & g, std::vector< int > & room, int index)`

Adds the cell index to the vector if the cell is active and deactivates the cell.

Parameters

<i>g</i>	BitGrid in which to evaluate the cell index
<i>room</i>	In-Out parameter where the index is added if the cell is active
<i>index</i>	Index of the cell to be evaluated

Returns

True if index is added and the cell is deactivated in the initial grid
 False for invalid initial index or inactive cell; no index added

6.4.2.5 `bool inverseGrow (BitGrid & g, std::vector< int > & segment, int index)`

Adds the cell index to the vector if the cell is inactive and activates the cell.

Parameters

<i>g</i>	BitGrid in which to evaluate the cell index
<i>room</i>	In-Out parameter where the index is added if the cell is inactive
<i>index</i>	Index of the cell to be evaluated

Returns

True if index is added and the cell is activated in the initial grid
 False for invalid initial index or active cell; no index added

6.4.2.6 `void inverseRegionGrowing (BitGrid & initialGrid, std::vector< BitGrid > & segList, std::vector< BitGrid > & boundaries, double minArea)`

Search for the inactive segments with minimal area in the given grid.

Parameters

<i>g</i>	BitGrid in which to search for the segments
<i>segList</i>	Return parameter storing the segments found
<i>boundaries</i>	Return parameter storing the neighbouring cells of the segments found
<i>minArea</i>	Minimal area of the calculated segment

6.4.2.7 `void regionGrowing (BitGrid & g, std::vector< BitGrid > & segList, double minArea, int cellStep = 1)`

Search for the active segments with minimal area in the given grid.

Parameters

<i>g</i>	BitGrid in which to search for the segments
<i>segList</i>	Return parameter storing the segments found

Parameters

<i>minArea</i>	Minimal area of the calculated segment
<i>cellStep</i>	Specifies how many cells around to search for the active cells (default = 1)

6.4.2.8 void regionGrowingAround (BitGrid & *g*, std::vector< std::vector< int >> *trialsVec*, std::vector< BitGrid > & *segList*)

Search for the region of active cells in the given grid around cells given by the trial vectors.

Parameters

<i>g</i>	BitGrid in which to search for the region
<i>trialsVec</i>	Vector of the vectors of the indexes around (+-1 cell) which to search for the region
<i>segList</i>	Return parameter storing the segments found

Index

- [/home/edita/spcl2/src/MathUtils.h, 92](#)
- [/home/edita/spcl2/src/RANSAC.h, 100](#)
- [/home/edita/spcl2/src/RegionGrowing.h, 101](#)
- [/home/edita/spcl2/src/filesystem_utils.h, 91](#)
- add
 - [BitGrid, 10, 11](#)
- addFloor
 - [Project, 66](#)
- addOpening
 - [Project, 67](#)
- addPlane
 - [Project, 67](#)
 - [Room, 75](#)
 - [Wall, 86](#)
- addPoint
 - [Downsampler, 21](#)
 - [Histogram, 29](#)
 - [Polygon, 63](#)
- addPointCloud
 - [Opening, 31](#)
 - [Plane, 38](#)
 - [Project, 67](#)
 - [Room, 75](#)
- addRoom
 - [Floor, 24](#)
 - [Project, 67](#)
 - [Wall, 86](#)
- addWall
 - [Floor, 24](#)
 - [Project, 67](#)
- angleBetweenPlanes
 - [MathUtils.h, 94](#)
- averageD
 - [MathUtils.h, 94](#)
- BitGrid, 9
 - [add, 10, 11](#)
 - [BitGrid, 10](#)
 - [get, 11, 12](#)
 - [getActiveCells, 12](#)
 - [getActiveCellsIndex, 13](#)
 - [getArea, 13](#)
 - [getBBox, 13](#)
 - [getBoundingPolygon, 13](#)
 - [getDimensions, 13](#)
 - [getPoint, 13](#)
 - [getPoints, 14](#)
 - [getSize, 14](#)
 - [getSizeX, 14](#)
 - [getSizeY, 14](#)
 - [getSizeZ, 14](#)
 - [getStepX, 14](#)
 - [getStepY, 15](#)
 - [getStepZ, 15](#)
 - [getX, 15](#)
 - [getY, 15](#)
 - [pointExists, 16](#)
 - [remove, 16](#)
 - [toSVG, 16](#)
- boundaryGrowing2
 - [RegionGrowing.h, 102](#)
- BoundingBoxInfo, 16
 - [giveBoundingBoxMax, 17](#)
 - [giveBoundingBoxMin, 17](#)
 - [range, 17](#)
- Cell, 18
 - [hash_value, 19](#)
 - [operator==, 18](#)
- checkLocalMatch
 - [Plane, 39](#)
- checkMatchingArea
 - [Plane, 39](#)
- computeHistogram
 - [Histogram, 29](#)
- createProjectFile
 - [Project, 68](#)
- cross
 - [MathUtils.h, 95](#)
- cubicEq
 - [MathUtils.h, 95](#)
- det2
 - [MathUtils.h, 95](#)
- det3
 - [MathUtils.h, 96](#)
- directory_exists
 - [filesystem_utils.h, 91](#)
- distanceBetweenPlanes
 - [Plane, 39](#)
- distanceTo
 - [Plane, 40](#)
- dot
 - [MathUtils.h, 96](#)
- downsample
 - [Downsampler, 21](#)
- Downsampler, 19
 - [addPoint, 21](#)
 - [downsample, 21](#)

- Downsampler, [20](#), [21](#)
- getIndex, [21](#)
- getPoints, [22](#)
- getSize, [22](#)
- isActive, [22](#)
- setSizes, [22](#)
- setStep, [23](#)
- eig3
 - MathUtils.h, [96](#)
- filesystem_utils.h
 - directory_exists, [91](#)
- findPlanes
 - RANSAC.h, [100](#)
- Floor, [23](#)
 - addRoom, [24](#)
 - addWall, [24](#)
 - getCeilingLevel, [25](#)
 - getFloorLevel, [25](#)
 - getId, [25](#)
 - getMax, [25](#)
 - getMin, [25](#)
 - getPoints, [25](#)
 - getRoomIDs, [26](#)
 - getRooms, [26](#)
 - getWallIDs, [26](#)
 - getWalls, [26](#)
 - setCeilingLevel, [27](#)
 - setFloorLevel, [27](#)
 - setId, [27](#)
 - setMax, [27](#)
 - setMin, [27](#)
 - setRoomIDs, [28](#)
 - setWallIDs, [28](#)
- generateOuterPlane
 - Plane, [41](#)
- get
 - BitGrid, [11](#), [12](#)
- getActiveCells
 - BitGrid, [12](#)
- getActiveCellsIndex
 - BitGrid, [13](#)
- getArea
 - BitGrid, [13](#)
 - Room, [75](#)
- getBBox
 - BitGrid, [13](#)
- getBoundingPolygon
 - BitGrid, [13](#)
- getCeilingLevel
 - Floor, [25](#)
 - Room, [75](#)
- getCoords
 - Point, [46](#)
- getCornersX
 - Opening, [32](#)
- getCornersY
 - Opening, [32](#)
- getCornersZ
 - Opening, [32](#)
- getDimensions
 - BitGrid, [13](#)
 - Opening, [32](#)
- getFilepath
 - PointCloud, [48](#)
- getFloor
 - Project, [68](#)
- getFloorId
 - Plane, [41](#)
 - Wall, [86](#)
- getFloorLevel
 - Floor, [25](#)
 - Room, [75](#)
- getFloors
 - Project, [68](#)
- getFolder
 - Project, [68](#)
- getHistogram
 - Histogram, [30](#)
- getId
 - Floor, [25](#)
 - Opening, [32](#)
 - Plane, [41](#)
 - Point, [46](#)
 - PointCloud, [48](#)
 - Room, [75](#)
 - Wall, [86](#)
- getIndex
 - Downsampler, [21](#)
- getLocalMatch
 - Plane, [41](#)
- getMax
 - Floor, [25](#)
 - Plane, [42](#)
 - Room, [76](#)
- getMin
 - Floor, [25](#)
 - Plane, [42](#)
 - Room, [76](#)
- getName
 - PointCloud, [48](#)
- getNormal
 - Plane, [42](#)
 - PointNormal, [54](#)
- getNormalVector
 - Plane, [42](#)
- getNumberOfPoints
 - PointCloud, [48](#)
- getNx
 - PointNormal, [54](#)
- getNy
 - PointNormal, [54](#)
- getNz
 - PointNormal, [54](#)
- getOpening

- Project, 68
- getOpeningType
 - Opening, 33
- getOpenings
 - Project, 69
- getPlane
 - Project, 69
- getPlaneIDs
 - Room, 76
 - Wall, 86
- getPlanes
 - Project, 69
 - Room, 76
 - Wall, 87
- getPoint
 - BitGrid, 13
- getPointCloud
 - Project, 69
- getPointCloudIDs
 - Opening, 33
 - Plane, 43
 - Room, 76
- getPointClouds
 - Opening, 33
 - Plane, 43
 - Project, 69
 - Room, 77
- getPointType
 - PointCloud, 49
- getPoints
 - BitGrid, 14
 - Downsampler, 22
 - Floor, 25
 - Opening, 33
 - Plane, 43
 - PointCloud, 48
 - Room, 77
 - Wall, 87
- getRoom
 - Project, 70
- getRoomIDs
 - Floor, 26
 - Wall, 87
- getRooms
 - Floor, 26
 - Project, 70
 - Wall, 87, 88
- getSize
 - BitGrid, 14
 - Downsampler, 22
- getSizeX
 - BitGrid, 14
- getSizeY
 - BitGrid, 14
- getSizeZ
 - BitGrid, 14
- getStepX
 - BitGrid, 14
- getStepY
 - BitGrid, 15
- getStepZ
 - BitGrid, 15
- getTranslation
 - Project, 70
- getTrials
 - RegionGrowing.h, 103
- getTrials2
 - RegionGrowing.h, 103
- getWall
 - Project, 70
- getWallIDs
 - Floor, 26
- getWalls
 - Floor, 26
 - Project, 71
- getD
 - Plane, 41
- getX
 - BitGrid, 15
 - PointXYZ, 59
 - PointXY, 56
- getY
 - BitGrid, 15
 - PointXYZ, 59
 - PointXY, 57
- getZ
 - PointXYZ, 59
- giveAverageCoord
 - MathUtils.h, 96
- giveBoundingBoxMax
 - BoundingBoxInfo, 17
- giveBoundingBoxMin
 - BoundingBoxInfo, 17
- giveCenterPlane
 - Plane, 43
- giveLocalCS
 - MathUtils.h, 97
- giveLtoGVectorsFromLocalCS
 - MathUtils.h, 97
- grow
 - RegionGrowing.h, 103
- hash_value
 - Cell, 19
- Histogram, 28
 - addPoint, 29
 - computeHistogram, 29
 - getHistogram, 30
 - Histogram, 29
- HistogramInfo, 30
- import
 - Project, 71
- inverseGrow
 - RegionGrowing.h, 104
- inverseRegionGrowing
 - RegionGrowing.h, 104

- isActive
 - Downsampler, 22
- isInnerPlane
 - Plane, 44
- isOuter
 - Wall, 88
- isPointInside
 - Polygon, 64
- length
 - Vector3, 81
- length_sqr
 - Vector3, 81
- makeWall
 - Plane, 44
- MathUtils.h
 - angleBetweenPlanes, 94
 - averageD, 94
 - cross, 95
 - cubicEq, 95
 - det2, 95
 - det3, 96
 - dot, 96
 - eig3, 96
 - giveAverageCoord, 96
 - giveLocalCS, 97
 - giveLtoGVectorsFromLocalCS, 97
 - moveBackCSorigin, 97
 - moveCSorigin, 98
 - normalize, 98
 - planeFromPoints, 98
 - projectPointsToXYPlane, 99
 - rotatePoint, 99
 - rotatePoints, 99
 - rotatePointsToLocal, 99
- moveBackCSorigin
 - MathUtils.h, 97
- moveCSorigin
 - MathUtils.h, 98
- normalize
 - MathUtils.h, 98
 - Vector3, 81
- normalized
 - Vector3, 81
- openProjectFile
 - Project, 71
- Opening, 31
 - addPointCloud, 31
 - getCornersX, 32
 - getCornersY, 32
 - getCornersZ, 32
 - getDimensions, 32
 - getId, 32
 - getOpeningType, 33
 - getPointCloudIDs, 33
 - getPointClouds, 33
 - getPoints, 33
 - setCornersX, 34
 - setCornersY, 34
 - setCornersZ, 34
 - setDimensions, 34
 - setId, 34
 - setOpeningType, 35
 - setPointCloudIDs, 35
- operator*
 - Vector3, 82
- operator*=
 - Vector3, 82
- operator^
 - Vector3, 84
- operator^=
 - Vector3, 85
- operator+
 - Vector3, 82
- operator+=
 - Vector3, 83
- operator-
 - Vector3, 83
- operator-=
 - Vector3, 83
- operator/
 - Vector3, 84
- operator/=
 - Vector3, 84
- operator==
 - Cell, 18
- PTXWriter, 73
 - write, 73
- Plane, 35
 - addPointCloud, 38
 - checkLocalMatch, 39
 - checkMatchingArea, 39
 - distanceBetweenPlanes, 39
 - distanceTo, 40
 - generateOuterPlane, 41
 - getFloorId, 41
 - getId, 41
 - getLocalMatch, 41
 - getMax, 42
 - getMin, 42
 - getNormal, 42
 - getNormalVector, 42
 - getPointCloudIDs, 43
 - getPointClouds, 43
 - getPoints, 43
 - getD, 41
 - giveCenterPlane, 43
 - isInnerPlane, 44
 - makeWall, 44
 - Plane, 37, 38
 - setFloorId, 45
 - setId, 45
 - setMax, 45
 - setMin, 45

- setNormal, 45
- setD, 44
- plane_parameters
 - RANSAC.h, 101
- planeFromPoints
 - MathUtils.h, 98
- Point
 - getCoords, 46
 - getId, 46
 - setId, 46
- Point< Dimensions >, 46
- PointCloud, 47
 - getFilepath, 48
 - getId, 48
 - getName, 48
 - getNumberOfPoints, 48
 - getPointType, 49
 - getPoints, 48
 - PointCloud, 48
 - readXYZNormalPoint, 49
 - readXYZPoint, 49
 - setFilepath, 50
 - setId, 50
 - setName, 50
 - setPointType, 50
 - setPoints, 50
 - writePointsCount, 51
 - writeXYZNormalPoint, 51
 - writeXYZPoint, 51
- pointExists
 - BitGrid, 16
- PointNormal, 52
 - getNormal, 54
 - getNx, 54
 - getNy, 54
 - getNz, 54
 - PointNormal, 53, 54
 - setNx, 54
 - setNy, 55
 - setNz, 55
- PointXYZInfo, 61
- PointXYZNormal, 61
 - PointXYZNormal, 62
- PointXYZNormalInfo, 63
- PointXYZ, 58
 - getX, 59
 - getY, 59
 - getZ, 59
 - PointXYZ, 59
 - setCoords, 60
 - setX, 60
 - setY, 60
 - setZ, 60
- PointXY, 55
 - getX, 56
 - getY, 57
 - PointXY, 56
 - setX, 57
 - setY, 57
- Polygon, 63
 - addPoint, 63
 - isPointInside, 64
- Project, 64
 - addFloor, 66
 - addOpening, 67
 - addPlane, 67
 - addPointCloud, 67
 - addRoom, 67
 - addWall, 67
 - createProjectFile, 68
 - getFloor, 68
 - getFloors, 68
 - getFolder, 68
 - getOpening, 68
 - getOpenings, 69
 - getPlane, 69
 - getPlanes, 69
 - getPointCloud, 69
 - getPointClouds, 69
 - getRoom, 70
 - getRooms, 70
 - getTranslation, 70
 - getWall, 70
 - getWalls, 71
 - import, 71
 - openProjectFile, 71
 - Project, 66
 - setTranslation, 71
 - update, 71, 72
- projectPointsToXYPlane
 - MathUtils.h, 99
- RANSAC.h
 - findPlanes, 100
 - plane_parameters, 101
 - random_three_pnt, 101
- random_three_pnt
 - RANSAC.h, 101
- range
 - BoundingBoxInfo, 17
- readXYZNormalPoint
 - PointCloud, 49
- readXYZPoint
 - PointCloud, 49
- regionGrowing
 - RegionGrowing.h, 104
- RegionGrowing.h
 - boundaryGrowing2, 102
 - getTrials, 103
 - getTrials2, 103
 - grow, 103
 - inverseGrow, 104
 - inverseRegionGrowing, 104
 - regionGrowing, 104
 - regionGrowingAround, 105
- regionGrowingAround
 - RegionGrowing.h, 105

- remove
 - BitGrid, 16
- Room, 74
 - addPlane, 75
 - addPointCloud, 75
 - getArea, 75
 - getCeilingLevel, 75
 - getFloorLevel, 75
 - getId, 75
 - getMax, 76
 - getMin, 76
 - getPlaneIDs, 76
 - getPlanes, 76
 - getPointCloudIDs, 76
 - getPointClouds, 77
 - getPoints, 77
 - setArea, 77
 - setCeilingLevel, 77
 - setFloorLevel, 78
 - setId, 78
 - setMax, 78
 - setMin, 78
 - setPlaneIDs, 78
 - setPointCloudIDs, 79
- rotatePoint
 - MathUtils.h, 99
- rotatePoints
 - MathUtils.h, 99
- rotatePointsToLocal
 - MathUtils.h, 99
- setArea
 - Room, 77
- setCeilingLevel
 - Floor, 27
 - Room, 77
- setCoords
 - PointXYZ, 60
- setCornersX
 - Opening, 34
- setCornersY
 - Opening, 34
- setCornersZ
 - Opening, 34
- setDimensions
 - Opening, 34
- setFilepath
 - PointCloud, 50
- setFloorId
 - Plane, 45
 - Wall, 88
- setFloorLevel
 - Floor, 27
 - Room, 78
- setId
 - Floor, 27
 - Opening, 34
 - Plane, 45
 - Point, 46
 - PointCloud, 50
 - Room, 78
 - Wall, 88
- setMax
 - Floor, 27
 - Plane, 45
 - Room, 78
- setMin
 - Floor, 27
 - Plane, 45
 - Room, 78
- setName
 - PointCloud, 50
- setNormal
 - Plane, 45
- setNx
 - PointNormal, 54
- setNy
 - PointNormal, 55
- setNz
 - PointNormal, 55
- setOpeningType
 - Opening, 35
- setOuter
 - Wall, 88
- setPlaneIDs
 - Room, 78
 - Wall, 89
- setPointCloudIDs
 - Opening, 35
 - Room, 79
- setPointType
 - PointCloud, 50
- setPoints
 - PointCloud, 50
- setRoomIDs
 - Floor, 28
 - Wall, 89
- setSizes
 - Downsampler, 22
- setStep
 - Downsampler, 23
- setTranslation
 - Project, 71
- setWallIDs
 - Floor, 28
- setD
 - Plane, 44
- setX
 - PointXYZ, 60
 - PointXY, 57
- setY
 - PointXYZ, 60
 - PointXY, 57
- setZ
 - PointXYZ, 60
- toSVG
 - BitGrid, 16

- update
 - Project, [71](#), [72](#)
- Vector3, [79](#)
 - length, [81](#)
 - length_sqr, [81](#)
 - normalize, [81](#)
 - normalized, [81](#)
 - operator*, [82](#)
 - operator*=[82](#)
 - operator[^], [84](#)
 - operator[^]=, [85](#)
 - operator+, [82](#)
 - operator+=, [83](#)
 - operator-, [83](#)
 - operator-=, [83](#)
 - operator/, [84](#)
 - operator/=, [84](#)
 - Vector3, [80](#), [81](#)
- Wall, [85](#)
 - addPlane, [86](#)
 - addRoom, [86](#)
 - getFloorId, [86](#)
 - getId, [86](#)
 - getPlaneIDs, [86](#)
 - getPlanes, [87](#)
 - getPoints, [87](#)
 - getRoomIDs, [87](#)
 - getRooms, [87](#), [88](#)
 - isOuter, [88](#)
 - setFloorId, [88](#)
 - setId, [88](#)
 - setOuter, [88](#)
 - setPlaneIDs, [89](#)
 - setRoomIDs, [89](#)
- write
 - PTXWriter, [73](#)
- writePointsCount
 - PointCloud, [51](#)
- writeXYZNormalPoint
 - PointCloud, [51](#)
- writeXYZPoint
 - PointCloud, [51](#)