VM2D

1.5

Создано системой Doxygen 1.8.13

Оглавление

1 Вихревые методы для решения двумерных задач 1

2 Список задач 3

3 Алфавитный указатель групп 5

3.1 Группы 5

4 Алфавитный указатель пространств имен 7

4.1 Пространства имен 7

5 Иерархический список классов 9

5.1 Иерархия классов 9

6 Алфавитный указатель классов 11

6.1 Классы 11

7 Список файлов 13

7.1 Файлы 13

8 Группы 17

8.1 Параллельные функции 17

8.1.1 Подробное описание 18

8.1.2 Функции 18

8.1.2.1 CalcConvVeloToSetOfPoints() 18

8.1.2.2 CalcDiffVeloI1I2ToSetOfPoints() 19

8.1.2.3 GetConvVelocityToSetOfPoints() 19

8.1.2.4 GetConvVelocityToSetOfPointsFromVirtualVortexes() 20

8.1.2.5 GetDiffVelocityI0I3ToSetOfPointsAndViscousStresses() 21

8.1.2.6 GetWakeInfluence() 22

8.1.2.7 Inside() 23

8.1.2.8 VirtualWakeSynchronize() 25

8.1.2.9 WakeSynchronize() 25

9 Пространства имен 27
9.1 Пространство имен defaults . . . 27
9.1.1 Подробное описание . . . 28
9.1.2 Функции . . . 28
9.1.2.1 defaultAirfoil() . . . 29
9.1.2.2 defaultFileAirfoil() . . . 29
9.1.2.3 defaultFileSource() . . . 29
9.1.2.4 defaultFileWake() . . . 30
9.1.3 Переменные . . . 30
9.1.3.1 defaultAirfoilsDir . . . 30
9.1.3.2 defaultAngle . . . 30
9.1.3.3 defaultBasePoint . . . 30
9.1.3.4 defaultBoundaryCondition . . . 31
9.1.3.5 defaultCopyPspFile . . . 31
9.1.3.6 defaultDelta . . . 31
9.1.3.7 defaultDistFar . . . 31
9.1.3.8 defaultInverse . . . 31
9.1.3.9 defaultMechanicalSystem . . . 32
9.1.3.10 defaultMechanicalSystemType . . . 32
9.1.3.11 defaultNameLength . . . 32
9.1.3.12 defaultNp . . . 32
9.1.3.13 defaultPanelsType . . . 32
9.1.3.14 defaultPspFile . . . 33
9.1.3.15 defaultQueueLogStream . . . 33
9.1.3.16 defaultSaveTXT . . . 33
9.1.3.17 defaultSaveVP . . . 33
9.1.3.18 defaultSaveVTK . . . 33
9.1.3.19 defaultScale . . . 34
9.1.3.20 defaultTimeAccel . . . 34
9.1.3.21 defaultTimeStart . . . 34
9.1.3.22 defaultVortexPerPanel . . . 34
9.1.3.23 defaultVRef . . . 34
9.1.3.24 defaultWakesDir . . . 34
9.1.3.25 defaultWorld2DLogStream . . . 34

10 Классы 35

10.1 Класс Airfoil 35

10.1.1 Подробное описание 37

10.1.2 Конструктор(ы) 38

10.1.2.1 Airfoil() [1/2] 38

10.1.2.2 Airfoil() [2/2] 38

10.1.2.3 ~Airfoil() 39

10.1.3 Методы 39

10.1.3.1 CalcNrmTauLen() 39

10.1.3.2 GetGabarits() 41

10.1.3.3 IFCUDA() [1/6] 42

10.1.3.4 IFCUDA() [2/6] 42

10.1.3.5 IFCUDA() [3/6] 42

10.1.3.6 IFCUDA() [4/6] 42

10.1.3.7 IFCUDA() [5/6] 42

10.1.3.8 IFCUDA() [6/6] 42

10.1.3.9 isAfter() 42

10.1.3.10 isInsideGabarits() 43

10.1.3.11 isOutsideGabarits() 44

10.1.3.12 IsPointInAirfoil() 45

10.1.3.13 Move() 46

10.1.3.14 ReadFromFile() 47

10.1.3.15 Rotate() 48

10.1.3.16 Scale() 49

10.1.4 Данные класса 50

10.1.4.1 gammaThrough 50

10.1.4.2 inverse 50

10.1.4.3 J 51

10.1.4.4 len 51

10.1.4.5 lowLeft 51

10.1.4.6 m 51
10.1.4.7 np 51
10.1.4.8 nrm 52
10.1.4.9 numberInPassport 52
10.1.4.10 phiAfl 52
10.1.4.11 r 52
10.1.4.12 rcm 52
10.1.4.13 tau 53
10.1.4.14 upRight 53
10.1.4.15 v 53
10.1.4.16 viscousStress 53
10.1.4.17 W 54
10.2 Структура AirfoilParams 54
10.2.1 Подробное описание 55
10.2.2 Данные класса 55
10.2.2.1 angle 55
10.2.2.2 basePoint 56
10.2.2.3 boundaryCondition 56
10.2.2.4 fileAirfoil 56
10.2.2.5 inverse 56
10.2.2.6 mechanicalSystem 56
10.2.2.7 mechanicalSystemParameters 57
10.2.2.8 mechanicalSystemType 57
10.2.2.9 panelsType 57
10.2.2.10 scale 57
10.3 Класс AirfoilRect 58
10.3.1 Подробное описание 60
10.3.2 Конструктор(ы) 60
10.3.2.1 AirfoilRect() [1/2] 61
10.3.2.2 AirfoilRect() [2/2] 61

10.3.2.3 ~AirfoilRect() 61
10.3.3 Методы 62
10.3.3.1 CalcNrmTauLen() 62
10.3.3.2 GetDiffVelocityI0I3ToSetOfPointsAndViscousStresses() 63
10.3.3.3 GetGabarits() 66
10.3.3.4 IFCUDA() [1/6] 67
10.3.3.5 IFCUDA() [2/6] 68
10.3.3.6 IFCUDA() [3/6] 68
10.3.3.7 IFCUDA() [4/6] 68
10.3.3.8 IFCUDA() [5/6] 68
10.3.3.9 IFCUDA() [6/6] 68
10.3.3.10 isAfter() 68
10.3.3.11 isInsideGabarits() 69
10.3.3.12 isOutsideGabarits() 70
10.3.3.13 IsPointInAirfoil() 71
10.3.3.14 Move() 72
10.3.3.15 ReadFromFile() 73
10.3.3.16 Rotate() 75
10.3.3.17 Scale() 76
10.3.4 Данные класса 77
10.3.4.1 gammaThrough 77
10.3.4.2 inverse 77
10.3.4.3 J 77
10.3.4.4 len 78
10.3.4.5 lowLeft 78
10.3.4.6 m 78
10.3.4.7 np 78
10.3.4.8 nrm 78
10.3.4.9 numberInPassport 79
10.3.4.10 phiAfl 79

10.3.4.11 r 79
10.3.4.12 rcm 79
10.3.4.13 tau 79
10.3.4.14 upRight 80
10.3.4.15 v 80
10.3.4.16 viscousStress 80
10.3.4.17 W 80
10.4 Класс Boundary 81
10.4.1 Подробное описание 83
10.4.2 Конструктор(ы) 83
10.4.2.1 Boundary() 83
10.4.2.2 ~Boundary() 84
10.4.3 Методы 85
10.4.3.1 ComputeAttachedSheetsIntensity() 85
10.4.3.2 FillMatrixFromOther() 86
10.4.3.3 FillMatrixSelf() 87
10.4.3.4 FillRhs() 87
10.4.3.5 FillRhsFromOther() 88
10.4.3.6 GetConvVelocityAtVirtualVortexes() 89
10.4.3.7 GetUnknownsSize() 90
10.4.3.8 SolutionToFreeVortexSheetAndVirtualVortex() 90
10.4.4 Данные класса 91
10.4.4.1 afl 91
10.4.4.2 CC 91
10.4.4.3 numberInPassport 91
10.4.4.4 oldSheets 92
10.4.4.5 sheetDim 92
10.4.4.6 sheets 92
10.4.4.7 virtualWake 92
10.4.4.8 vortexPerPanel 93

10.4.4.9 W . . . 93
10.5 Класс BoundaryConstLayerAver . . . 93
10.5.1 Подробное описание . . . 95
10.5.2 Конструктор(ы) . . . 95
10.5.2.1 BoundaryConstLayerAver() . . . 95
10.5.2.2 ~BoundaryConstLayerAver() . . . 96
10.5.3 Методы . . . 97
10.5.3.1 Alpha() . . . 97
10.5.3.2 ComputeAttachedSheetsIntensity() . . . 98
10.5.3.3 FillMatrixFromOther() . . . 98
10.5.3.4 FillMatrixSelf() . . . 100
10.5.3.5 FillRhs() . . . 102
10.5.3.6 FillRhsFromOther() . . . 104
10.5.3.7 GetConvVelocityAtVirtualVortexes() . . . 106
10.5.3.8 GetConvVelocityToSetOfPoints() . . . 107
10.5.3.9 GetConvVelocityToSetOfPointsFromVirtualVortexes() . . . 109
10.5.3.10 GetUnknownsSize() . . . 112
10.5.3.11 GetWakeInfluence() . . . 112
10.5.3.12 Lambda() . . . 114
10.5.3.13 Omega() . . . 115
10.5.3.14 SolutionToFreeVortexSheetAndVirtualVortex() . . . 116
10.5.4 Данные класса . . . 118
10.5.4.1 afl . . . 118
10.5.4.2 CC . . . 118
10.5.4.3 numberInPassport . . . 118
10.5.4.4 oldSheets . . . 119
10.5.4.5 sheetDim . . . 119
10.5.4.6 sheets . . . 119
10.5.4.7 virtualWake . . . 119
10.5.4.8 vortexPerPanel . . . 120

10.5.4.9 W . . . 120
10.6 Класс Gpu . . . 120
10.6.1 Подробное описание . . . 121
10.6.2 Конструктор(ы) . . . 121
10.6.2.1 Gpu() . . . 121
10.6.2.2 ~Gpu() . . . 122
10.6.3 Методы . . . 124
10.6.3.1 setAccelCoeff() . . . 124
10.6.3.2 setCollapseCoeff() . . . 125
10.6.4 Данные класса . . . 125
10.6.4.1 W . . . 126
10.7 Класс LogStream . . . 126
10.7.1 Подробное описание . . . 127
10.7.2 Конструктор(ы) . . . 127
10.7.2.1 LogStream() . . . 128
10.7.2.2 ~LogStream() . . . 128
10.7.3 Методы . . . 128
10.7.3.1 assignStream() . . . 128
10.7.3.2 endl() . . . 129
10.7.3.3 inheritStream() . . . 129
10.7.3.4 operator()() . . . 130
10.7.4 Данные класса . . . 131
10.7.4.1 label . . . 131
10.7.4.2 pStr . . . 132
10.8 Класс MeasureVP . . . 132
10.8.1 Подробное описание . . . 133
10.8.2 Конструктор(ы) . . . 133
10.8.2.1 MeasureVP() . . . 133
10.8.2.2 ~MeasureVP() . . . 134
10.8.3 Методы . . . 135

10.8.3.1 CalcSaveVP() 135
10.8.3.2 ReadPointsFromFile() 142
10.8.4 Данные класса 143
10.8.4.1 additionalWake 143
10.8.4.2 domainRadius 143
10.8.4.3 historyPoints 144
10.8.4.4 initialPoints 144
10.8.4.5 pressure 144
10.8.4.6 velocity 144
10.8.4.7 W 144
10.9 Класс Mechanics 145
10.9.1 Подробное описание 147
10.9.2 Конструктор(ы) 148
10.9.2.1 Mechanics() 148
10.9.2.2 ~Mechanics() 149
10.9.3 Методы 150
10.9.3.1 FillAtt() 150
10.9.3.2 FillMechanicsRhs() 150
10.9.3.3 FillMechanicsRowsAndCross() 151
10.9.3.4 GenerateForcesHeader() 152
10.9.3.5 GenerateForcesString() 153
10.9.3.6 GeneratePositionHeader() 154
10.9.3.7 GeneratePositionString() 155
10.9.3.8 GetHydroDynamForce() 156
10.9.3.9 Move() 156
10.9.3.10 PositionOfAirfoilRcm() 156
10.9.3.11 ReadParametersFromDictionary() 157
10.9.3.12 ReadSpecificParametersFromDictionary() 158
10.9.3.13 SolutionToMechanicalSystem() 158
10.9.3.14 VeloOfAirfoilPanels() 159

10.9.3.15 VeloOfAirfoilRcm() . . . 159
10.9.4 Данные класса . . . 160
10.9.4.1 afl . . . 160
10.9.4.2 boundary . . . 160
10.9.4.3 degOfFreedom . . . 160
10.9.4.4 hydroDynamForce . . . 161
10.9.4.5 hydroDynamMoment . . . 161
10.9.4.6 isDeform . . . 161
10.9.4.7 isMoves . . . 161
10.9.4.8 isRotate . . . 161
10.9.4.9 mechParamsParser . . . 162
10.9.4.10 numberInPassport . . . 162
10.9.4.11 Phi . . . 162
10.9.4.12 Phi0 . . . 162
10.9.4.13 PhiOld . . . 162
10.9.4.14 Rcm . . . 162
10.9.4.15 Rcm0 . . . 163
10.9.4.16 RcmOld . . . 163
10.9.4.17 Vcm . . . 163
10.9.4.18 Vcm0 . . . 163
10.9.4.19 VcmOld . . . 163
10.9.4.20 virtVortParams . . . 164
10.9.4.21 W . . . 164
10.9.4.22 Wcm . . . 164
10.9.4.23 Wcm0 . . . 164
10.9.4.24 WcmOld . . . 164
10.10 Класс MechanicsRigidGivenLaw . . . 165
10.10.1 Подробное описание . . . 167
10.10.2 Конструктор(ы) . . . 167
10.10.2.1 MechanicsRigidGivenLaw() . . . 167

10.10.2.2 ~MechanicsRigidGivenLaw() . . . 168
10.10.3 Методы . . . 169
10.10.3.1 FillAtt() . . . 169
10.10.3.2 FillMechanicsRhs() . . . 169
10.10.3.3 FillMechanicsRowsAndCross() . . . 170
10.10.3.4 GenerateForcesHeader() . . . 171
10.10.3.5 GenerateForcesString() . . . 172
10.10.3.6 GeneratePositionHeader() . . . 173
10.10.3.7 GeneratePositionString() . . . 174
10.10.3.8 GetHydroDynamForce() . . . 175
10.10.3.9 Move() . . . 176
10.10.3.10 PositionOfAirfoilRcm() . . . 177
10.10.3.11 ReadSpecificParametersFromDictionary() . . . 177
10.10.3.12 SolutionToMechanicalSystem() . . . 178
10.10.3.13 VeloOfAirfoilPanels() . . . 178
10.10.3.14 VeloOfAirfoilRcm() . . . 179
10.10.4 Данные класса . . . 180
10.10.4.1 afl . . . 180
10.10.4.2 boundary . . . 180
10.10.4.3 degOfFreedom . . . 180
10.10.4.4 hydroDynamForce . . . 181
10.10.4.5 hydroDynamMoment . . . 181
10.10.4.6 isDeform . . . 181
10.10.4.7 isMoves . . . 181
10.10.4.8 isRotate . . . 181
10.10.4.9 mechParamsParser . . . 182
10.10.4.10 numberInPassport . . . 182
10.10.4.11 Phi . . . 182
10.10.4.12 Phi0 . . . 182
10.10.4.13 PhiOld . . . 182

10.10.4.14 Rcm . . . 182
10.10.4.15 Rcm0 . . . 183
10.10.4.16 RcmOld . . . 183
10.10.4.17 Vcm . . . 183
10.10.4.18 Vcm0 . . . 183
10.10.4.19 VcmOld . . . 183
10.10.4.20 virtVortParams . . . 184
10.10.4.21 W . . . 184
10.10.4.22 Wcm . . . 184
10.10.4.23 Wcm0 . . . 184
10.10.4.24 WcmOld . . . 184
10.11 Класс MechanicsRigidImmovable . . . 185
10.11.1 Подробное описание . . . 187
10.11.2 Конструктор(ы) . . . 187
10.11.2.1 MechanicsRigidImmovable() . . . 187
10.11.2.2 ~MechanicsRigidImmovable() . . . 188
10.11.3 Методы . . . 189
10.11.3.1 ComputeDisplacementRcm() . . . 189
10.11.3.2 FillAtt() . . . 189
10.11.3.3 FillMechanicsRhs() . . . 190
10.11.3.4 FillMechanicsRowsAndCross() . . . 190
10.11.3.5 GenerateForcesHeader() . . . 191
10.11.3.6 GenerateForcesString() . . . 192
10.11.3.7 GeneratePositionHeader() . . . 193
10.11.3.8 GeneratePositionString() . . . 194
10.11.3.9 GetHydroDynamForce() . . . 195
10.11.3.10 Move() . . . 196
10.11.3.11 PositionOfAirfoilRcm() . . . 196
10.11.3.12 ReadSpecificParametersFromDictionary() . . . 197
10.11.3.13 SolutionToMechanicalSystem() . . . 197

10.11.3.14 VeloOfAirfoilPanels() . . . 198
10.11.3.15 VeloOfAirfoilRcm() . . . 199
10.11.4 Данные класса . . . 199
10.11.4.1 afl . . . 199
10.11.4.2 boundary . . . 200
10.11.4.3 degOfFreedom . . . 200
10.11.4.4 hydroDynamForce . . . 200
10.11.4.5 hydroDynamMoment . . . 200
10.11.4.6 isDeform . . . 200
10.11.4.7 isMoves . . . 201
10.11.4.8 isRotate . . . 201
10.11.4.9 mechParamsParser . . . 201
10.11.4.10 numberInPassport . . . 201
10.11.4.11 Phi . . . 201
10.11.4.12 Phi0 . . . 202
10.11.4.13 PhiOld . . . 202
10.11.4.14 Rcm . . . 202
10.11.4.15 Rcm0 . . . 202
10.11.4.16 RcmOld . . . 202
10.11.4.17 Vcm . . . 203
10.11.4.18 Vcm0 . . . 203
10.11.4.19 VcmOld . . . 203
10.11.4.20 virtVortParams . . . 203
10.11.4.21 W . . . 203
10.11.4.22 Wcm . . . 204
10.11.4.23 Wcm0 . . . 204
10.11.4.24 WcmOld . . . 204
10.12 Класс MechanicsRigidOscillMon . . . 204
10.12.1 Подробное описание . . . 206
10.12.2 Конструктор(ы) . . . 207

10.12.2.1 MechanicsRigidOscillMon() 207
10.12.2.2 ~MechanicsRigidOscillMon() 207
10.12.3 Методы 208
10.12.3.1 FillAtt() 208
10.12.3.2 FillMechanicsRhs() 209
10.12.3.3 FillMechanicsRowsAndCross() 210
10.12.3.4 GenerateForcesHeader() 211
10.12.3.5 GenerateForcesString() 212
10.12.3.6 GeneratePositionHeader() 213
10.12.3.7 GeneratePositionString() 214
10.12.3.8 GetHydroDynamForce() 215
10.12.3.9 Move() 216
10.12.3.10 PositionOfAirfoilRcm() 217
10.12.3.11 ReadSpecificParametersFromDictionary() 218
10.12.3.12 SolutionToMechanicalSystem() 218
10.12.3.13 VeloOfAirfoilPanels() 219
10.12.3.14 VeloOfAirfoilRcm() 220
10.12.4 Данные класса 221
10.12.4.1 afl 221
10.12.4.2 b 221
10.12.4.3 boundary 221
10.12.4.4 degOfFreedom 222
10.12.4.5 hydroDynamForce 222
10.12.4.6 hydroDynamMoment 222
10.12.4.7 isDeform 222
10.12.4.8 isMoves 222
10.12.4.9 isRotate 223
10.12.4.10 k 223
10.12.4.11 m 223
10.12.4.12 mechParamsParser 223

10.12.4.13 numberInPassport . . . 223
10.12.4.14 Phi . . . 223
10.12.4.15 Phi0 . . . 224
10.12.4.16 PhiOld . . . 224
10.12.4.17 Rcm . . . 224
10.12.4.18 Rcm0 . . . 224
10.12.4.19 RcmOld . . . 224
10.12.4.20 Vcm . . . 225
10.12.4.21 Vcm0 . . . 225
10.12.4.22 VcmOld . . . 225
10.12.4.23 virtVortParams . . . 225
10.12.4.24 W . . . 225
10.12.4.25 Wcm . . . 226
10.12.4.26 Wcm0 . . . 226
10.12.4.27 WcmOld . . . 226
10.13 Класс MechanicsRigidOscillPart . . . 226
10.13.1 Подробное описание . . . 229
10.13.2 Конструктор(ы) . . . 229
10.13.2.1 MechanicsRigidOscillPart() . . . 229
10.13.2.2 ~MechanicsRigidOscillPart() . . . 230
10.13.3 Методы . . . 230
10.13.3.1 FillAtt() . . . 230
10.13.3.2 FillMechanicsRhs() . . . 231
10.13.3.3 FillMechanicsRowsAndCross() . . . 232
10.13.3.4 GenerateForcesHeader() . . . 232
10.13.3.5 GenerateForcesString() . . . 233
10.13.3.6 GeneratePositionHeader() . . . 234
10.13.3.7 GeneratePositionString() . . . 235
10.13.3.8 GetHydroDynamForce() . . . 236
10.13.3.9 Move() . . . 238

10.13.3.10 PositionOfAirfoilRcm() 239
10.13.3.11 ReadSpecificParametersFromDictionary() 239
10.13.3.12 SolutionToMechanicalSystem() 240
10.13.3.13 VeloOfAirfoilPanels() 241
10.13.3.14 VeloOfAirfoilRcm() 242
10.13.4 Данные класса 242
10.13.4.1 afl 243
10.13.4.2 b 243
10.13.4.3 boundary 243
10.13.4.4 degOfFreedom 243
10.13.4.5 hydroDynamForce 243
10.13.4.6 hydroDynamMoment 244
10.13.4.7 isDeform 244
10.13.4.8 isMoves 244
10.13.4.9 isRotate 244
10.13.4.10 k 244
10.13.4.11 m 245
10.13.4.12 mechParamsParser 245
10.13.4.13 numberInPassport 245
10.13.4.14 Phi 245
10.13.4.15 Phi0 245
10.13.4.16 PhiOld 245
10.13.4.17 Rcm 246
10.13.4.18 Rcm0 246
10.13.4.19 RcmOld 246
10.13.4.20 n 246
10.13.4.21 $n0$ 246
10.13.4.22 nOld 247
10.13.4.23 Vcm 247
10.13.4.24 Vcm0 247

10.13.4.25 VcmOld . . . 247
10.13.4.26 virtVortParams . . . 247
10.13.4.27 W . . . 248
10.13.4.28 Wcm . . . 248
10.13.4.29 Wcm0 . . . 248
10.13.4.30 WcmOld . . . 248
10.13.4.31 y . . . 248
10.13.4.32 y0 . . . 248
10.13.4.33 yOld . . . 249
10.14 Класс MechanicsRigidRotateMon . . . 249
10.14.1 Подробное описание . . . 252
10.14.2 Конструктор(ы) . . . 252
10.14.2.1 MechanicsRigidRotateMon() . . . 252
10.14.2.2 ~MechanicsRigidRotateMon() . . . 253
10.14.3 Методы . . . 253
10.14.3.1 Alpha() . . . 253
10.14.3.2 FillAtt() . . . 254
10.14.3.3 FillMechanicsRhs() . . . 256
10.14.3.4 FillMechanicsRowsAndCross() . . . 258
10.14.3.5 GenerateForcesHeader() . . . 259
10.14.3.6 GenerateForcesString() . . . 260
10.14.3.7 GeneratePositionHeader() . . . 261
10.14.3.8 GeneratePositionString() . . . 262
10.14.3.9 GetHydroDynamForce() . . . 263
10.14.3.10 GetMz() . . . 264
10.14.3.11 Lambda() . . . 265
10.14.3.12 Move() . . . 266
10.14.3.13 PositionOfAirfoilRcm() . . . 266
10.14.3.14 ReadSpecificParametersFromDictionary() . . . 267
10.14.3.15 SolutionToMechanicalSystem() . . . 268

10.14.3.16 VeloOfAirfoilPanels() . . . 269
10.14.3.17 VeloOfAirfoilRcm() . . . 269
10.14.4 Данные класса . . . 270
10.14.4.1 afl . . . 270
10.14.4.2 b . . . 270
10.14.4.3 boundary . . . 271
10.14.4.4 degOfFreedom . . . 271
10.14.4.5 hydroDynamForce . . . 271
10.14.4.6 hydroDynamMoment . . . 271
10.14.4.7 isDeform . . . 271
10.14.4.8 isMoves . . . 272
10.14.4.9 isRotate . . . 272
10.14.4.10 J . . . 272
10.14.4.11 k . . . 272
10.14.4.12 mechParamsParser . . . 272
10.14.4.13 Mz . . . 273
10.14.4.14 numberInPassport . . . 273
10.14.4.15 Phi . . . 273
10.14.4.16 Phi0 . . . 273
10.14.4.17 PhiOld . . . 273
10.14.4.18 Rcm . . . 273
10.14.4.19 Rcm0 . . . 274
10.14.4.20 RcmOld . . . 274
10.14.4.21 tMomentAccel . . . 274
10.14.4.22 tRotateAccel . . . 274
10.14.4.23 Vcm . . . 274
10.14.4.24 Vcm0 . . . 275
10.14.4.25 VcmOld . . . 275
10.14.4.26 virtVortParams . . . 275
10.14.4.27 W . . . 275

10.14.4.28 Wcm . . . 275
10.14.4.29 Wcm0 . . . 276
10.14.4.30 WcmOld . . . 276
10.15 Структура NumericalSchemes . . . 276
10.15.1 Подробное описание . . . 276
10.15.2 Данные класса . . . 277
10.15.2.1 linearSystemSolver . . . 277
10.15.2.2 velocityComputation . . . 277
10.15.2.3 wakeMotionIntegrator . . . 277
10.16 Шаблон класса nummatrix< T, n, m > . . . 277
10.16.1 Подробное описание . . . 279
10.16.2 Конструктор(ы) . . . 280
10.16.2.1 nummatrix() [1/7] . . . 280
10.16.2.2 nummatrix() [2/7] . . . 280
10.16.2.3 nummatrix() [3/7] . . . 281
10.16.2.4 nummatrix() [4/7] . . . 281
10.16.2.5 nummatrix() [5/7] . . . 283
10.16.2.6 nummatrix() [6/7] . . . 284
10.16.2.7 nummatrix() [7/7] . . . 285
10.16.3 Методы . . . 285
10.16.3.1 norm1() . . . 285
10.16.3.2 norminf() . . . 286
10.16.3.3 operator &() [1/2] . . . 286
10.16.3.4 operator &() [2/2] . . . 287
10.16.3.5 operator std::vector< std::vector< T >>() . . . 288
10.16.3.6 operator"!=() . . . 288
10.16.3.7 operator∗() . . . 289
10.16.3.8 operator∗=() . . . 290
10.16.3.9 operator+() . . . 290
10.16.3.10 operator+=() . . . 291

10.16.3.11 operator-() [1/2] . . . 292
10.16.3.12 operator-() [2/2] . . . 292
10.16.3.13 operator-=() . . . 293
10.16.3.14 operator/=() . . . 294
10.16.3.15 operator==() . . . 295
10.16.3.16 operator[]() [1/2] . . . 296
10.16.3.17 operator[]() [2/2] . . . 296
10.16.3.18 size() . . . 297
10.16.3.19 toIdentity() . . . 297
10.16.3.20 toZero() . . . 298
10.16.3.21 transpose() . . . 299
10.16.4 Данные класса . . . 299
10.16.4.1 r . . . 299
10.17 Шаблон класса numvector< T, n > . . . 299
10.17.1 Подробное описание . . . 302
10.17.2 Конструктор(ы) . . . 303
10.17.2.1 numvector() [1/8] . . . 303
10.17.2.2 numvector() [2/8] . . . 303
10.17.2.3 numvector() [3/8] . . . 304
10.17.2.4 numvector() [4/8] . . . 304
10.17.2.5 numvector() [5/8] . . . 305
10.17.2.6 numvector() [6/8] . . . 306
10.17.2.7 numvector() [7/8] . . . 306
10.17.2.8 numvector() [8/8] . . . 307
10.17.3 Методы . . . 307
10.17.3.1 dist2To() . . . 308
10.17.3.2 distTo() . . . 308
10.17.3.3 kcross() . . . 309
10.17.3.4 length() . . . 310
10.17.3.5 length2() . . . 311

10.17.3.6 member() 311
10.17.3.7 norm1() 312
10.17.3.8 normalize() 312
10.17.3.9 norminf() 313
10.17.3.10 operator &() 314
10.17.3.11 operator std::set< P >() 314
10.17.3.12 operator std::vector< P >() 315
10.17.3.13 operator"!=() 315
10.17.3.14 operator∗() 316
10.17.3.15 operator∗=() 317
10.17.3.16 operator+() [1/2] 317
10.17.3.17 operator+() [2/2] 318
10.17.3.18 operator+=() 318
10.17.3.19 operator-() [1/2] 319
10.17.3.20 operator-() [2/2] 320
10.17.3.21 operator-=() 320
10.17.3.22 operator/=() 321
10.17.3.23 operator=() 322
10.17.3.24 operator==() 322
10.17.3.25 operator[]() [1/2] 323
10.17.3.26 operator[]() [2/2] 324
10.17.3.27 operator^() [1/2] 325
10.17.3.28 operator^() [2/2] 325
10.17.3.29 rotateLeft() 326
10.17.3.30 size() 327
10.17.3.31 toSet() 327
10.17.3.32 toVector() 327
10.17.3.33 toZero() 328
10.17.3.34 unit() 329
10.17.4 Данные класса 330

10.17.4.1 r . . . 330
10.18 Класс Parallel . . . 330
10.18.1 Подробное описание . . . 331
10.18.2 Методы . . . 331
10.18.2.1 SplitMPI() . . . 331
10.18.2.2 SplitMPIone() . . . 332
10.18.3 Данные класса . . . 333
10.18.3.1 commWork . . . 333
10.18.3.2 myidWork . . . 334
10.18.3.3 nProcWork . . . 334
10.19 Структура parProp . . . 334
10.19.1 Подробное описание . . . 335
10.19.2 Данные класса . . . 335
10.19.2.1 disp . . . 335
10.19.2.2 len . . . 336
10.19.2.3 myDisp . . . 336
10.19.2.4 myLen . . . 336
10.19.2.5 totalLen . . . 336
10.20 Класс Passport . . . 337
10.20.1 Подробное описание . . . 338
10.20.2 Конструктор(ы) . . . 338
10.20.2.1 Passport() . . . 338
10.20.2.2 ~Passport() . . . 340
10.20.3 Методы . . . 340
10.20.3.1 GetAllParamsFromParser() . . . 340
10.20.3.2 PrintAllParams() . . . 343
10.20.4 Данные класса . . . 344
10.20.4.1 airfoilParams . . . 345
10.20.4.2 airfoilsDir . . . 345
10.20.4.3 dir . . . 345

10.20.4.4 info . . . 345
10.20.4.5 numericalSchemes . . . 345
10.20.4.6 physicalProperties . . . 346
10.20.4.7 problemName . . . 346
10.20.4.8 problemNumber . . . 346
10.20.4.9 timeDiscretizationProperties . . . 346
10.20.4.10 wakeDiscretizationProperties . . . 346
10.20.4.11 wakesDir . . . 347
10.21 Структура PhysicalProperties . . . 347
10.21.1 Подробное описание . . . 348
10.21.2 Методы . . . 348
10.21.2.1 accelCft() . . . 349
10.21.2.2 addCurrTime() . . . 349
10.21.2.3 getCurrTime() . . . 350
10.21.2.4 setCurrTime() . . . 350
10.21.2.5 V0() . . . 351
10.21.3 Данные класса . . . 351
10.21.3.1 currTime . . . 351
10.21.3.2 nu . . . 352
10.21.3.3 rho . . . 352
10.21.3.4 timeAccel . . . 352
10.21.3.5 vInf . . . 352
10.21.3.6 vRef . . . 352
10.22 Класс Point2D . . . 353
10.22.1 Подробное описание . . . 356
10.22.2 Конструктор(ы) . . . 356
10.22.2.1 Point2D() [1/4] . . . 356
10.22.2.2 Point2D() [2/4] . . . 356
10.22.2.3 Point2D() [3/4] . . . 357
10.22.2.4 Point2D() [4/4] . . . 357

10.22.2.5 ~Point2D() . . . 358
10.22.3 Методы . . . 358
10.22.3.1 CreateMpiType() . . . 359
10.22.3.2 dist2To() . . . 359
10.22.3.3 distTo() . . . 360
10.22.3.4 kcross() . . . 361
10.22.3.5 length() . . . 361
10.22.3.6 length2() . . . 362
10.22.3.7 member() . . . 362
10.22.3.8 norm1() . . . 363
10.22.3.9 normalize() . . . 363
10.22.3.10 norminf() . . . 364
10.22.3.11 operator std::set< P >() . . . 364
10.22.3.12 operator std::vector< P >() . . . 364
10.22.3.13 operator"!=() . . . 365
10.22.3.14 operator&() . . . 366
10.22.3.15 operator*() . . . 366
10.22.3.16 operator*=() . . . 367
10.22.3.17 operator+() [1/2] . . . 367
10.22.3.18 operator+() [2/2] . . . 368
10.22.3.19 operator+=() . . . 369
10.22.3.20 operator-() [1/2] . . . 370
10.22.3.21 operator-() [2/2] . . . 371
10.22.3.22 operator-=() . . . 371
10.22.3.23 operator/=() . . . 372
10.22.3.24 operator==() . . . 372
10.22.3.25 operator[]() [1/2] . . . 373
10.22.3.26 operator[]() [2/2] . . . 374
10.22.3.27 operator^() [1/2] . . . 374
10.22.3.28 operator^() [2/2] . . . 375

10.22.3.29 rotated() . . . 375
10.22.3.30 rotateLeft() . . . 376
10.22.3.31 size() . . . 377
10.22.3.32 toSet() . . . 377
10.22.3.33 toVector() . . . 378
10.22.3.34 toZero() . . . 378
10.22.3.35 unit() . . . 379
10.22.4 Данные класса . . . 379
10.22.4.1 mpiPoint2D . . . 379
10.22.4.2 r . . . 380
10.23 Класс Preprocessor . . . 380
10.23.1 Подробное описание . . . 382
10.23.2 Конструктор(ы) . . . 383
10.23.2.1 Preprocessor() . . . 383
10.23.2.2 ~Preprocessor() . . . 384
10.23.3 Методы . . . 384
10.23.3.1 afterSlashParser() . . . 384
10.23.3.2 inCharParser() . . . 385
10.23.3.3 inInlineCommentParser() . . . 386
10.23.3.4 inMultilineCommentParser() . . . 387
10.23.3.5 inStringParser() . . . 388
10.23.3.6 normalParser() . . . 389
10.23.3.7 processSymbol() . . . 391
10.23.4 Данные класса . . . 391
10.23.4.1 currentParser . . . 392
10.23.4.2 inCharLastEscape . . . 392
10.23.4.3 inInlineLastEscape . . . 392
10.23.4.4 initialInput . . . 392
10.23.4.5 inMultilineLastStar . . . 392
10.23.4.6 inStringLastEscape . . . 393

10.23.4.7 intermediateOutput . . . 393
10.23.4.8 resultStream . . . 393
10.23.4.9 resultString . . . 393
10.24 Класс Queue . . . 394
10.24.1 Подробное описание . . . 396
10.24.2 Конструктор(ы) . . . 396
10.24.2.1 Queue() . . . 396
10.24.2.2 ~Queue() . . . 397
10.24.3 Методы . . . 397
10.24.3.1 AddTask() . . . 397
10.24.3.2 ConstructProcStateVar() . . . 398
10.24.3.3 LoadTasksList() . . . 399
10.24.3.4 RunConveyer() . . . 402
10.24.3.5 TaskSplit() . . . 402
10.24.3.6 TaskUpdate() . . . 406
10.24.4 Данные класса . . . 408
10.24.4.1 commSolving . . . 408
10.24.4.2 commStarting . . . 408
10.24.4.3 currentKvant . . . 409
10.24.4.4 finished . . . 409
10.24.4.5 flagFinish . . . 409
10.24.4.6 groupAll . . . 409
10.24.4.7 groupSolving . . . 410
10.24.4.8 groupStarting . . . 410
10.24.4.9 info . . . 410
10.24.4.10 kvantTime . . . 410
10.24.4.11 myidAll . . . 411
10.24.4.12 myProcState . . . 411
10.24.4.13 myProcStateVar . . . 411
10.24.4.14 nextKvant . . . 412

10.24.4.15 nProcAll . . . 412
10.24.4.16 numberOfTask . . . 412
10.24.4.17 parallel . . . 412
10.24.4.18 prepared . . . 413
10.24.4.19 procState . . . 413
10.24.4.20 procStateVar . . . 413
10.24.4.21 sizeCommSolving . . . 414
10.24.4.22 solving . . . 414
10.24.4.23 task . . . 414
10.24.4.24 world2D . . . 414
10.25 Класс Sheet . . . 415
10.25.1 Подробное описание . . . 416
10.25.2 Конструктор(ы) . . . 416
10.25.2.1 Sheet() . . . 416
10.25.2.2 ~Sheet() . . . 416
10.25.3 Методы . . . 417
10.25.3.1 SetLayersDim() . . . 417
10.25.4 Данные класса . . . 417
10.25.4.1 attachedSourceSheet . . . 418
10.25.4.2 attachedVortexSheet . . . 418
10.25.4.3 freeVortexSheet . . . 418
10.26 Класс StreamParser . . . 418
10.26.1 Подробное описание . . . 421
10.26.2 Конструктор(ы) . . . 421
10.26.2.1 StreamParser() [1/3] . . . 421
10.26.2.2 StreamParser() [2/3] . . . 422
10.26.2.3 StreamParser() [3/3] . . . 423
10.26.2.4 ~StreamParser() . . . 424
10.26.3 Методы . . . 424
10.26.3.1 get() [1/6] . . . 424

10.26.3.2 get() [2/6] . . . 425
10.26.3.3 get() [3/6] . . . 426
10.26.3.4 get() [4/6] . . . 427
10.26.3.5 get() [5/6] . . . 429
10.26.3.6 get() [6/6] . . . 430
10.26.3.7 ParseStream() . . . 431
10.26.3.8 ReplaceVarsInString() . . . 432
10.26.3.9 SetDefault() . . . 434
10.26.3.10 SplitString() . . . 435
10.26.3.11 StringToVector() . . . 436
10.26.3.12 UpperCase() . . . 437
10.26.3.13 VectorStringToString() . . . 438
10.26.4 Данные класса . . . 439
10.26.4.1 database . . . 439
10.26.4.2 defaults . . . 439
10.26.4.3 info . . . 440
10.26.4.4 switchers . . . 440
10.26.4.5 vars . . . 440
10.27 Класс Task . . . 440
10.27.1 Подробное описание . . . 441
10.27.2 Конструктор(ы) . . . 441
10.27.2.1 Task() . . . 441
10.27.2.2 ~Task() . . . 442
10.27.3 Данные класса . . . 442
10.27.3.1 nProc . . . 442
10.27.3.2 passport . . . 442
10.27.3.3 proc . . . 443
10.27.3.4 startEndKvant . . . 443
10.27.3.5 state . . . 443
10.27.3.6 wTime . . . 443

10.28 Структура TimeDiscretizationProperties . . . 443
10.28.1 Подробное описание . . . 444
10.28.2 Данные класса . . . 444
10.28.2.1 dt . . . 444
10.28.2.2 nameLength . . . 445
10.28.2.3 saveTXT . . . 445
10.28.2.4 saveVP . . . 445
10.28.2.5 saveVTK . . . 445
10.28.2.6 timeStart . . . 445
10.28.2.7 timeStop . . . 446
10.29 Класс Times . . . 446
10.29.1 Подробное описание . . . 447
10.29.2 Конструктор(ы) . . . 448
10.29.2.1 Times() . . . 448
10.29.2.2 ~Times() . . . 448
10.29.3 Методы . . . 449
10.29.3.1 dT() . . . 449
10.29.3.2 GenerateStatHeader() . . . 449
10.29.3.3 GenerateStatString() . . . 450
10.29.3.4 ToZero() [1/2] . . . 451
10.29.3.5 ToZero() [2/2] . . . 452
10.29.4 Данные класса . . . 453
10.29.4.1 timeCalcVortexConvVelo . . . 453
10.29.4.2 timeCalcVortexDiffVelo . . . 453
10.29.4.3 timeCheckInside . . . 453
10.29.4.4 timeFillMatrixAndRhs . . . 453
10.29.4.5 timeGetHydroDynamForce . . . 453
10.29.4.6 timeMoveVortexes . . . 454
10.29.4.7 timeOther . . . 454
10.29.4.8 timeReserveMemoryForMatrixAndRhs . . . 454

10.29.4.9 timeRestruct . . . 454
10.29.4.10 timeSaveKadr . . . 454
10.29.4.11 timeSolveLinearSystem . . . 455
10.29.4.12 timeVP . . . 455
10.29.4.13 timeWakeSort . . . 455
10.29.4.14 timeWholeStep . . . 455
10.29.4.15 W . . . 455
10.30 Класс Velocity . . . 456
10.30.1 Подробное описание . . . 457
10.30.2 Конструктор(ы) . . . 457
10.30.2.1 Velocity() . . . 457
10.30.2.2 ~Velocity() . . . 458
10.30.3 Методы . . . 458
10.30.3.1 CalcConvVelo() . . . 458
10.30.3.2 CalcDiffVelo() . . . 459
10.30.3.3 CalcDiffVeloI1I2ToSetOfPointsFromPanels() . . . 461
10.30.4 Данные класса . . . 462
10.30.4.1 virtualVortexesParams . . . 462
10.30.4.2 W . . . 462
10.30.4.3 wakeVortexesParams . . . 462
10.31 Класс VelocityBiotSavart . . . 463
10.31.1 Подробное описание . . . 464
10.31.2 Конструктор(ы) . . . 464
10.31.2.1 VelocityBiotSavart() . . . 464
10.31.2.2 ~VelocityBiotSavart() . . . 465
10.31.3 Методы . . . 465
10.31.3.1 CalcConvVelo() . . . 465
10.31.3.2 CalcConvVeloToSetOfPoints() . . . 466
10.31.3.3 CalcDiffVelo() . . . 469
10.31.3.4 CalcDiffVeloI1I2ToSetOfPoints() . . . 471

10.31.3.5 CalcDiffVeloI1I2ToSetOfPointsFromPanels() . . . 474
10.31.4 Данные класса . . . 476
10.31.4.1 virtualVortexesParams . . . 476
10.31.4.2 W . . . 476
10.31.4.3 wakeVortexesParams . . . 477
10.32 Класс VirtualWake . . . 477
10.32.1 Подробное описание . . . 478
10.32.2 Конструктор(ы) . . . 478
10.32.2.1 VirtualWake() . . . 478
10.32.2.2 ~VirtualWake() . . . 479
10.32.3 Методы . . . 479
10.32.3.1 ReadFromFile() . . . 479
10.32.4 Данные класса . . . 480
10.32.4.1 aflPan . . . 480
10.32.4.2 bnd . . . 481
10.32.4.3 vecHalfGamma . . . 481
10.32.4.4 vtx . . . 481
10.32.4.5 W . . . 481
10.33 Класс Vortex2D . . . 482
10.33.1 Подробное описание . . . 483
10.33.2 Конструктор(ы) . . . 483
10.33.2.1 Vortex2D() [1/2] . . . 483
10.33.2.2 Vortex2D() [2/2] . . . 484
10.33.2.3 ~Vortex2D() . . . 484
10.33.3 Методы . . . 484
10.33.3.1 CreateMpiType() . . . 484
10.33.3.2 g() [1/2] . . . 485
10.33.3.3 g() [2/2] . . . 486
10.33.3.4 r() [1/2] . . . 486
10.33.3.5 r() [2/2] . . . 487

10.33.4 Данные класса . . . 487
10.33.4.1 gam . . . 487
10.33.4.2 mpiVortex2D . . . 488
10.33.4.3 offsGam . . . 488
10.33.4.4 offsPos . . . 488
10.33.4.5 pos . . . 488
10.34 Структура VortexesParams . . . 489
10.34.1 Подробное описание . . . 490
10.34.2 Данные класса . . . 490
10.34.2.1 convVelo . . . 490
10.34.2.2 diffVelo . . . 490
10.34.2.3 epsastWake . . . 490
10.34.2.4 I0 . . . 491
10.34.2.5 I1 . . . 491
10.34.2.6 I2 . . . 491
10.34.2.7 I3 . . . 491
10.35 Класс Wake . . . 492
10.35.1 Подробное описание . . . 493
10.35.2 Конструктор(ы) . . . 494
10.35.2.1 Wake() . . . 494
10.35.2.2 ~Wake() . . . 494
10.35.3 Методы . . . 495
10.35.3.1 Collaps() . . . 495
10.35.3.2 GetPairs() . . . 497
10.35.3.3 MoveInside() . . . 499
10.35.3.4 MoveInsideMovingBoundary() . . . 501
10.35.3.5 ReadFromFile() . . . 503
10.35.3.6 RemoveFar() . . . 504
10.35.3.7 RemoveZero() . . . 505
10.35.3.8 Restruct() . . . 505

10.35.3.9 SaveKadr() 507
10.35.3.10 SaveKadrVtk() 508
10.35.4 Данные класса 510
10.35.4.1 collapseRightBorderParameter 510
10.35.4.2 collapseScaleParameter 510
10.35.4.3 neighb 511
10.35.4.4 vtx 511
10.35.4.5 W 511
10.36 Класс WakeDataBase 511
10.36.1 Подробное описание 512
10.36.2 Конструктор(ы) 512
10.36.2.1 WakeDataBase() 512
10.36.2.2 ~WakeDataBase() 513
10.36.3 Методы 513
10.36.3.1 ReadFromFile() 513
10.36.4 Данные класса 514
10.36.4.1 vtx 515
10.36.4.2 W 515
10.37 Структура WakeDiscretizationProperties 515
10.37.1 Подробное описание 516
10.37.2 Данные класса 516
10.37.2.1 delta 516
10.37.2.2 distFar 517
10.37.2.3 eps 517
10.37.2.4 eps2 517
10.37.2.5 epscol 517
10.37.2.6 fileSource 517
10.37.2.7 fileWake 518
10.37.2.8 vortexPerPanel 518
10.38 Класс World2D 518

10.38.1 Подробное описание . . . 521
10.38.2 Конструктор(ы) . . . 521
10.38.2.1 World2D() . . . 521
10.38.2.2 ~World2D() . . . 524
10.38.3 Методы . . . 524
10.38.3.1 CalcVortexVelo() . . . 524
10.38.3.2 CheckInside() . . . 530
10.38.3.3 FillMatrixAndRhs() . . . 531
10.38.3.4 GenerateMechanicsHeader() . . . 534
10.38.3.5 getAirfoil() . . . 535
10.38.3.6 getBoundary() . . . 535
10.38.3.7 getCuda() . . . 536
10.38.3.8 getCurrentStep() . . . 537
10.38.3.9 getInfo() [1/2] . . . 537
10.38.3.10 getInfo() [2/2] . . . 538
10.38.3.11 getNonConstAirfoil() . . . 539
10.38.3.12 getNonConstBoundary() . . . 539
10.38.3.13 getNonConstVelocity() . . . 540
10.38.3.14 getNumberOfAirfoil() . . . 541
10.38.3.15 getNumberOfBoundary() . . . 541
10.38.3.16 getParallel() . . . 542
10.38.3.17 getPassport() . . . 542
10.38.3.18 getSource() . . . 544
10.38.3.19 getTimestat() . . . 544
10.38.3.20 getVelocity() . . . 545
10.38.3.21 getWake() . . . 546
10.38.3.22 isFinished() . . . 546
10.38.3.23 MoveVortexes() . . . 547
10.38.3.24 ReserveMemoryForMatrixAndRhs() . . . 548
10.38.3.25 SolveLinearSystem() . . . 549

10.38.3.26 Step() 550

10.38.4 Данные класса 555

10.38.4.1 airfoil 555

10.38.4.2 boundary 556

10.38.4.3 cuda 556

10.38.4.4 currentStep 556

10.38.4.5 info 556

10.38.4.6 invMatr 556

10.38.4.7 matr 557

10.38.4.8 measureVP 557

10.38.4.9 mechanics 557

10.38.4.10 oldAirfoil 557

10.38.4.11 parallel 557

10.38.4.12 passport 558

10.38.4.13 rhs 558

10.38.4.14 sol 558

10.38.4.15 source 558

10.38.4.16 timestat 558

10.38.4.17 velocity 559

10.38.4.18 wake 559

11 Файлы 561

11.1 Файл Airfoil/Airfoil.cpp 561

11.1.1 Подробное описание 562

11.2 Файл Airfoil/Airfoil.h 562

11.2.1 Подробное описание 563

11.3 Файл Airfoil/AirfoilRect.cpp 563

11.3.1 Подробное описание 564

11.4 Файл Airfoil/AirfoilRect.h 564

11.4.1 Подробное описание 565

11.5 Файл Boundary/Boundary.cpp 565

11.5.1 Подробное описание 566

11.6 Файл Boundary/Boundary.h 566

11.6.1 Подробное описание 567

11.7 Файл Boundary/BoundaryConstLayerAver.cpp 567

11.7.1 Подробное описание 568

11.8 Файл Boundary/BoundaryConstLayerAver.h 568

11.8.1 Подробное описание 569

11.9 Файл defs/defs.cpp 569

11.9.1 Подробное описание 570

11.9.2 Функции 570

11.9.2.1 CurrentDataTime() 571

11.9.2.2 M4() 572

11.9.2.3 PrintHeaderToTextFile() 573

11.9.2.4 PrintLogoToTextFile() 573

11.9.2.5 SaveToStream() [1/4] 575

11.9.2.6 SaveToStream() [2/4] 576

11.9.2.7 SaveToStream() [3/4] 576

11.9.2.8 SaveToStream() [4/4] 577

11.9.3 Переменные 577

11.9.3.1 Months 577

11.10 Файл defs/defs.h . . . 577
11.10.1 Подробное описание . . . 580
11.10.2 Макросы . . . 581
11.10.2.1 DYN_SCHEDULE . . . 581
11.10.3 Типы . . . 581
11.10.3.1 timePeriod . . . 581
11.10.4 Функции . . . 581
11.10.4.1 boundDenom() . . . 581
11.10.4.2 CurrentDataTime() . . . 582
11.10.4.3 fileExistTest() . . . 583
11.10.4.4 fileNameStep() . . . 584
11.10.4.5 M4() . . . 584
11.10.4.6 operator<<() . . . 585
11.10.4.7 PrintHeaderToTextFile() . . . 586
11.10.4.8 PrintLogoToTextFile() . . . 587
11.10.4.9 SaveToStream() [1/4] . . . 588
11.10.4.10 SaveToStream() [2/4] . . . 589
11.10.4.11 SaveToStream() [3/4] . . . 590
11.10.4.12 SaveToStream() [4/4] . . . 590
11.10.4.13 sqr() . . . 590
11.10.5 Переменные . . . 591
11.10.5.1 DPI . . . 592
11.10.5.2 IDPI . . . 592
11.10.5.3 PI . . . 592
11.11 Файл Gpu/Gpu.cpp . . . 592
11.11.1 Подробное описание . . . 593
11.12 Файл Gpu/Gpu.h . . . 593
11.12.1 Подробное описание . . . 594
11.13 Файл Gpu/Gpudefs.h . . . 594
11.13.1 Подробное описание . . . 595

11.13.2 Макросы . . . 595
11.13.2.1 CU_CONV_TOBOU . . . 595
11.13.2.2 CU_CONV_TOWAKE . . . 595
11.13.2.3 CU_CONVVIRT . . . 595
11.13.2.4 CU_I0I3 . . . 596
11.13.2.5 CU_I1I2 . . . 596
11.13.2.6 CU_PAIRS . . . 596
11.13.2.7 CU_RHS . . . 596
11.13.2.8 CUBLOCK . . . 596
11.13.2.9 CUDA_DISABLE . . . 596
11.13.2.10 CUDA_ENABLE . . . 597
11.13.2.11 CUDAdef . . . 597
11.13.2.12 IFCUDA . . . 597
11.13.2.13 IFCUDA_ . . . 597
11.13.2.14 INC_VORT_DEV . . . 597
11.14 Файл LogStream/LogStream.cpp . . . 598
11.14.1 Подробное описание . . . 598
11.15 Файл LogStream/LogStream.h . . . 599
11.15.1 Подробное описание . . . 599
11.16 Файл MeasureVP/MeasureVP.cpp . . . 600
11.16.1 Подробное описание . . . 600
11.17 Файл MeasureVP/MeasureVP.h . . . 601
11.17.1 Подробное описание . . . 601
11.18 Файл Mechanics/Mechanics.cpp . . . 602
11.18.1 Подробное описание . . . 602
11.19 Файл Mechanics/Mechanics.h . . . 602
11.19.1 Подробное описание . . . 603
11.20 Файл Mechanics/MechanicsRigidGivenLaw.cpp . . . 604
11.20.1 Подробное описание . . . 604
11.21 Файл Mechanics/MechanicsRigidGivenLaw.h . . . 604

11.21.1 Подробное описание . . . 605
11.22 Файл Mechanics/MechanicsRigidImmovable.cpp . . . 606
11.22.1 Подробное описание . . . 606
11.23 Файл Mechanics/MechanicsRigidImmovable.h . . . 606
11.23.1 Подробное описание . . . 607
11.24 Файл Mechanics/MechanicsRigidOscillMon.cpp . . . 607
11.24.1 Подробное описание . . . 608
11.25 Файл Mechanics/MechanicsRigidOscillMon.h . . . 608
11.25.1 Подробное описание . . . 609
11.26 Файл Mechanics/MechanicsRigidOscillPart.cpp . . . 609
11.26.1 Подробное описание . . . 609
11.27 Файл Mechanics/MechanicsRigidOscillPart.h . . . 610
11.27.1 Подробное описание . . . 610
11.28 Файл Mechanics/MechanicsRigidRotateMon.cpp . . . 611
11.28.1 Подробное описание . . . 611
11.29 Файл Mechanics/MechanicsRigidRotateMon.h . . . 611
11.29.1 Подробное описание . . . 612
11.30 Файл numvector/nummatrix.h . . . 612
11.30.1 Подробное описание . . . 613
11.30.2 Функции . . . 613
11.30.2.1 operator &() . . . 613
11.30.2.2 operator<<() . . . 614
11.30.2.3 operator() . . . 615
11.31 Файл numvector/numvector.h . . . 615
11.31.1 Подробное описание . . . 618
11.31.2 Макросы . . . 618
11.31.2.1 DEPRECATED . . . 618
11.31.3 Функции . . . 619
11.31.3.1 cross() . . . 619
11.31.3.2 cross3() . . . 620

11.31.3.3 dist() . . . 621
11.31.3.4 dist2() . . . 622
11.31.3.5 dot() [1/2] . . . 623
11.31.3.6 dot() [2/2] . . . 624
11.31.3.7 KronProd() . . . 625
11.31.3.8 matDotVec() . . . 625
11.31.3.9 operator∗() [1/11] . . . 626
11.31.3.10 operator∗() [2/11] . . . 627
11.31.3.11 operator∗() [3/11] . . . 627
11.31.3.12 operator∗() [4/11] . . . 627
11.31.3.13 operator∗() [5/11] . . . 628
11.31.3.14 operator∗() [6/11] . . . 629
11.31.3.15 operator∗() [7/11] . . . 629
11.31.3.16 operator∗() [8/11] . . . 630
11.31.3.17 operator∗() [9/11] . . . 630
11.31.3.18 operator∗() [10/11] . . . 631
11.31.3.19 operator∗() [11/11] . . . 631
11.31.3.20 operator∗=() . . . 632
11.31.3.21 operator+() . . . 633
11.31.3.22 operator+=() . . . 633
11.31.3.23 operator<<() [1/2] . . . 633
11.31.3.24 operator<<() [2/2] . . . 634
11.31.3.25 Transpose() . . . 635
11.32 Файл Parallel/Parallel.cpp . . . 635
11.32.1 Подробное описание . . . 636
11.33 Файл Parallel/Parallel.h . . . 636
11.33.1 Подробное описание . . . 637
11.34 Файл Passport/Passport.cpp . . . 638
11.34.1 Подробное описание . . . 638
11.35 Файл Passport/Passport.h . . . 638

11.35.1 Подробное описание . . . 639
11.36 Файл Point2D/Point2D.cpp . . . 640
11.36.1 Подробное описание . . . 640
11.37 Файл Point2D/Point2D.h . . . 641
11.37.1 Подробное описание . . . 642
11.38 Файл Preprocessor/Preprocessor.cpp . . . 642
11.38.1 Подробное описание . . . 643
11.39 Файл Preprocessor/Preprocessor.h . . . 643
11.39.1 Подробное описание . . . 644
11.40 Файл Queue/Queue.cpp . . . 644
11.40.1 Подробное описание . . . 644
11.41 Файл Queue/Queue.h . . . 645
11.41.1 Подробное описание . . . 645
11.42 Файл Sheet/Sheet.cpp . . . 646
11.42.1 Подробное описание . . . 646
11.43 Файл Sheet/Sheet.h . . . 647
11.43.1 Подробное описание . . . 647
11.44 Файл StreamParser/StreamParser.cpp . . . 648
11.44.1 Подробное описание . . . 648
11.45 Файл StreamParser/StreamParser.h . . . 648
11.45.1 Подробное описание . . . 649
11.46 Файл Task/Task.h . . . 650
11.46.1 Подробное описание . . . 651
11.46.2 Перечисления . . . 651
11.46.2.1 TaskState . . . 651
11.47 Файл Times/Times.cpp . . . 652
11.47.1 Подробное описание . . . 652
11.48 Файл Times/Times.h . . . 653
11.48.1 Подробное описание . . . 653
11.49 Файл Velocity/Velocity.cpp . . . 654

11.49.1 Подробное описание . . . 654
11.50 Файл Velocity/Velocity.h . . . 654
11.50.1 Подробное описание . . . 655
11.51 Файл Velocity/VelocityBiotSavart.cpp . . . 656
11.51.1 Подробное описание . . . 656
11.51.2 Функции . . . 656
11.51.2.1 ModifyE2() . . . 657
11.52 Файл Velocity/VelocityBiotSavart.h . . . 657
11.52.1 Подробное описание . . . 658
11.53 Файл VirtualWake/VirtualWake.cpp . . . 659
11.53.1 Подробное описание . . . 659
11.54 Файл VirtualWake/VirtualWake.h . . . 659
11.54.1 Подробное описание . . . 660
11.55 Файл VM2D/VM2D.cpp . . . 661
11.55.1 Подробное описание . . . 661
11.55.2 Функции . . . 662
11.55.2.1 CreateMpiTypes() . . . 662
11.55.2.2 Initializers() . . . 662
11.55.2.3 main() . . . 663
11.56 Файл Vortex2D/Vortex2D.cpp . . . 664
11.56.1 Подробное описание . . . 665
11.57 Файл Vortex2D/Vortex2D.h . . . 665
11.57.1 Подробное описание . . . 666
11.57.2 Типы . . . 666
11.57.2.1 Source2D . . . 666
11.58 Файл Wake/Wake.cpp . . . 667
11.58.1 Подробное описание . . . 667
11.59 Файл Wake/Wake.h . . . 667
11.59.1 Подробное описание . . . 668
11.60 Файл WakeDataBase/WakeDataBase.cpp . . . 669
11.60.1 Подробное описание . . . 669
11.61 Файл WakeDataBase/WakeDataBase.h . . . 669
11.61.1 Подробное описание . . . 670
11.62 Файл World2D/World2D.cpp . . . 671
11.62.1 Подробное описание . . . 671
11.63 Файл World2D/World2D.h . . . 671
11.63.1 Подробное описание . . . 672

Алфавитный указатель 673

Глава 1

Вихревые методы для решения двумерных задач

Данный программный комплекс реализует вихревые методы решения двумерных задач гидродинамики и гидроупругости

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Дата

20 февраля 2019 г.

Версия

1.5

Глава 2

Список задач

Член Airfoil::Airfoil (const Airfoil &afl)

откомментировать — почему нельзя использовать default?

Член Boundary::FillMatrixFromOther (const Boundary &otherBoundary, Eigen::MatrixXd &matr)=0

Пока считается, что граничные условия одинаковые

Член Boundary::FillRhsFromOther (const Airfoil &otherAirfoil, Eigen::VectorXd &rhs)=0

Пока считается, что граничные условия одинаковые

Член BoundaryConstLayerAver::FillRhs (const Point2D &V0, Eigen::VectorXd &rhs, double *lastRhs, bool move, bool deform)

Это должно быть закомментировано или нет?

Член BoundaryConstLayerAver::GetConvVelocityToSetOfPointsFromVirtualVortexes (const WakeDataBase &pointsDb, std::vector< Point2D > &velo) const

Тут надо разобраться, как должно быть...

сделать if(move || deform)

Член MeasureVP::CalcSaveVP (const std::string &dir, size_t step, timePeriod &time)

Для успокоения добавить WakeSyncronize(), но после обучения его проверке hash

Пока передаем только средние значения свободного слоя на панелях

Член MeasureVP::W

Сделать учет давления на бесконечности

Член MechanicsRigidImmovable::FillMechanicsRowsAndCross (Eigen::MatrixXd &row, Eigen::MatrixXd &cross)

реализовать — не понял

Член MechanicsRigidOscillMon::FillMechanicsRowsAndCross (Eigen::MatrixXd &row, Eigen::MatrixXd &cross)

реализовать — не понял

Член MechanicsRigidOscillPart::FillMechanicsRowsAndCross (Eigen::MatrixXd &row, Eigen::MatrixXd &cross)

реализовать — не понял

Член VelocityBiotSavart::CalcDiffVeloI1I2ToSetOfPoints (const WakeDataBase &pointsDb, const std::vector< double > &domainRadius, const WakeDataBase &vorticesDb, std::vector< double > &I1, std::vector< Point2D > &I2)

Понять природу магической константы 8.0 и синхронизировать с GPU

Член VelocityBiotSavart::CalcDiffVeloI1I2ToSetOfPointsFromPanels (const WakeDataBase &pointsDb, const std::vector< double > &domainRadius, const Boundary &bnd, std::vector< double > &I1, std::vector< Point2D > &I2)

Понять природу магической константы 8.0 и синхронизировать с GPU

Сделать переменной и синхронизировать с GPU

Член Wake::GetPairs (int type)

Временно для профиля из 1000 панелей

Член Wake::MoveInsideMovingBoundary (const Point2D &newPos, const Point2D &oldPos, const Airfoil &oldAfl, const Airfoil &afl, size_t &panThrough)

сравнить производительности двух inside-ов

Член Wake::RemoveFar ()

Пока профиль 1, расстояние от его центра; сделать от самого правого профиля

Член World2D::FillMatrixAndRhs (timePeriod &time)

Влияние присоединенных слоев от других профилей пока выключено (для поступательного движения вроде не нужно)

Глава 3

Алфавитный указатель групп

3.1 Группы

Полный список групп.

Параллельные функции 17

Глава 4

Алфавитный указатель пространств имен

4.1 Пространства имен

Полный список пространств имен.

defaults
Глобальные параметры по умолчанию 27

Глава 5

Иерархический список классов

5.1 Иерархия классов

Иерархия классов.

Airfoil . . . 35
  AirfoilRect . . . 58
AirfoilParams . . . 54
Boundary . . . 81
  BoundaryConstLayerAver . . . 93
Gpu . . . 120
LogStream . . . 126
MeasureVP . . . 132
Mechanics . . . 145
  MechanicsRigidGivenLaw . . . 165
  MechanicsRigidImmovable . . . 185
  MechanicsRigidOscillMon . . . 204
  MechanicsRigidOscillPart . . . 226
  MechanicsRigidRotateMon . . . 249
NumericalSchemes . . . 276
nummatrix< T, n, m > . . . 277
numvector< T, n > . . . 299
numvector< double, 2 > . . . 299
  Point2D . . . 353
Parallel . . . 330
parProp . . . 334
Passport . . . 337
PhysicalProperties . . . 347
Preprocessor . . . 380
Queue . . . 394
Sheet . . . 415
StreamParser . . . 418
Task . . . 440
TimeDiscretizationProperties . . . 443
Times . . . 446
Velocity . . . 456
  VelocityBiotSavart . . . 463
Vortex2D . . . 482
VortexesParams . . . 489

WakeDataBase 511
VirtualWake 477
Wake 492
WakeDiscretizationProperties 515
World2D 518

Глава 6

Алфавитный указатель классов

6.1 Классы

Классы с их кратким описанием.

Airfoil
Абстрактный класс, определяющий обтекаемый профиль 35
AirfoilParams
Структура, задающая параметры профиля 54
AirfoilRect
Класс, определяющий тип обтекаемого профиля 58
Boundary
Абстрактный класс, определяющий способ удовлетворения граничного условия на обтекаемом профиле 81
BoundaryConstLayerAver
Класс, определяющий способ удовлетворения граничного условия на обтекаемом профиле 93
Gpu
Класс, обеспечивающий возможность выполнения вычислений на GPU по технологии Nvidia CUDA 120
LogStream
Класс, определяющий работу с потоком логов 126
MeasureVP
Класс, отвечающий за вычисление поля скорости и давления в заданых точках для вывода 132
Mechanics
Абстрактный класс, определяющий вид механической системы 145
MechanicsRigidGivenLaw
Класс, определяющий вид механической системы 165
MechanicsRigidImmovable
Класс, определяющий вид механической системы 185
MechanicsRigidOscillMon
Класс, определяющий вид механической системы 204
MechanicsRigidOscillPart
Класс, определяющий вид механической системы 226
MechanicsRigidRotateMon
Класс, определяющий вид механической системы 249
NumericalSchemes
Структура, задающая используемые численные схемы 276

numMatrix< T, n, m >
Шаблонный класс, определяющий матрицу фиксированного размера
Фактически представляет собой массив, для которого определено большое количество различных операций.
Для доступа к элементам матрицы используется оператор [][]
Оператор[] имитирует возврат ссылки/константной ссылки на numVector 277
numVector< T, n >
Шаблонный класс, определяющий вектор фиксированной длины
Фактически представляет собой массив, для которого определено большое количество различных операций.
Для доступа к элементам массива используется оператор [] 299
Parallel
Класс, опеделяющий параметры исполнения задачи в параллельном MPI-режиме 330
parProp
Стрктура, содержащая параметры исполнения задачи в параллельном MP↩
I-режиме 334
Passport
Класс, опеделяющий паспорт задачи 337
PhysicalProperties
Структура, задающая физические свойства задачи 347
Point2D
Класс, опеделяющий двумерный вектор 353
Preprocessor
Класс, позволяющий выполнять предварительную обработку файлов 380
Queue
Класс, опеделяющий список решаемых задач и очередь их прохождения 394
Sheet
Класс, опеделяющий слои на поверхности обтекаемого профиля 415
StreamParser
Класс, позволяющий выполнять разбор файлов и строк с настройками и параметрами 418
Task
Класс, опеделяющий описание каждой задачи в смысле прохождения процесса ее решения 440
TimeDiscretizationProperties
Структура, задающая параметры процесса интегрирования по времени 443
Times
Класс для сбора статистики времени исполнения основных шагов алгоритма и вывода ее в файл 446
Velocity
Абстрактный класс, определяющий способ вычисления скоростей 456
VelocityBiotSavart
Класс, определяющий способ вычисления скоростей 463
VirtualWake
Класс, опеделяющий вихревой след (пелену) 477
Vortex2D
Класс, опеделяющий двумерный вихревой элемент 482
VortexesParams
Структура, определяющая параметры виртуальных вихрей для отдельного профиля 489
Wake
Класс, опеделяющий вихревой след (пелену) 492
WakeDataBase 511
WakeDiscretizationProperties
Структура, задающая параметры параметры дискретизации вихревого следа . . 515
World2D
Класс, опеделяющий текущую решаемую задачу 518

Глава 7

Список файлов

7.1 Файлы

Полный список файлов.

Airfoil/Airfoil.cpp
Файл кода с описанием класса Airfoil 561
Airfoil/Airfoil.h
Заголовочный файл с описанием класса Airfoil 562
Airfoil/AirfoilRect.cpp
Файл кода с описанием класса AirfoilRect 563
Airfoil/AirfoilRect.h
Заголовочный файл с описанием класса AirfoilRect 564
Boundary/Boundary.cpp
Файл кода с описанием класса Boundary 565
Boundary/Boundary.h
Заголовочный файл с описанием класса Boundary 566
Boundary/BoundaryConstLayerAver.cpp
Файл кода с описанием класса BoundaryConstLayerAver 567
Boundary/BoundaryConstLayerAver.h
Заголовочный файл с описанием класса BoundaryConstLayerAver 568
defs/defs.cpp
Описание базовых вспомогательных функций 569
defs/defs.h
Описание базовых вспомогательных функций 577
Gpu/Gpu.cpp
Файл кода с описанием класса Gpu 592
Gpu/Gpu.h
Заголовочный файл с описанием класса Gpu 593
Gpu/Gpudefs.h
Описание констант и параметров для взаимодействия с графическим ускорителем 594
LogStream/LogStream.cpp
Файл кода с описанием класса LogStream 598
LogStream/LogStream.h
Заголовочный файл с описанием класса LogStream 599
MeasureVP/MeasureVP.cpp
Файл кода с описанием класса MeasureVP 600
MeasureVP/MeasureVP.h
Заголовочный файл с описанием класса MeasureVP 601
Mechanics/Mechanics.cpp
Файл кода с описанием класса Mechanics.cpp 602

Mechanics/Mechanics.h
Заголовочный файл с описанием класса Mechanics 602
Mechanics/MechanicsRigidGivenLaw.cpp
Файл кода с описанием класса MechanicsRigidGivenLaw 604
Mechanics/MechanicsRigidGivenLaw.h
Заголовочный файл с описанием класса MechanicsRigidGivenLaw 604
Mechanics/MechanicsRigidImmovable.cpp
Файл кода с описанием класса MechanicsRigidImmovable 606
Mechanics/MechanicsRigidImmovable.h
Заголовочный файл с описанием класса MechanicsRigidImmovable 606
Mechanics/MechanicsRigidOscillMon.cpp
Файл кода с описанием класса MechanicsRigidOscillMon 607
Mechanics/MechanicsRigidOscillMon.h
Заголовочный файл с описанием класса MechanicsRigidOscillMon 608
Mechanics/MechanicsRigidOscillPart.cpp
Файл кода с описанием класса MechanicsRigidOscillPart 609
Mechanics/MechanicsRigidOscillPart.h
Заголовочный файл с описанием класса MechanicsRigidOscillPart 610
Mechanics/MechanicsRigidRotateMon.cpp
Файл кода с описанием класса MechanicsRigidRotateMon 611
Mechanics/MechanicsRigidRotateMon.h
Заголовочный файл с описанием класса MechanicsRigidRotateMon 611
numvector/nummatrix.h
Описание класса nummatrix 612
numvector/numvector.h
Описание класса numvector 615
Parallel/Parallel.cpp
Файл кода с описанием класса Parallel 635
Parallel/Parallel.h
Заголовочный файл с описанием класса Parallel и структуры parProp 636
Passport/Passport.cpp
Файл кода с описанием класса Passport 638
Passport/Passport.h
Заголовочный файл с описанием класса Passport и соответствующими структурами 638
Point2D/Point2D.cpp
Файл кода с описанием класса Point2D 640
Point2D/Point2D.h
Заголовочный файл с описанием класса Point2D 641
Preprocessor/Preprocessor.cpp
Файл кода с описанием класса Preprocessor 642
Preprocessor/Preprocessor.h
Заголовочный файл с описанием класса Preprocessor 643
Queue/Queue.cpp
Файл кода с описанием класса Queue 644
Queue/Queue.h
Заголовочный файл с описанием класса Queue 645
Sheet/Sheet.cpp
Файл кода с описанием класса Sheet 646
Sheet/Sheet.h
Заголовочный файл с описанием класса Sheet 647
StreamParser/StreamParser.cpp
Файл кода с описанием класса StreamParser 648
StreamParser/StreamParser.h
Заголовочный файл с описанием класса StreamParser 648
Task/Task.h
Заголовочный файл с описанием класса Task 650

Times/Times.cpp
Файл кода с описанием класса Times 652
Times/Times.h
Заголовочный файл с описанием класса Times 653
Velocity/Velocity.cpp
Файл кода с описанием класса Velocity 654
Velocity/Velocity.h
Заголовочный файл с описанием класса Velocity 654
Velocity/VelocityBiotSavart.cpp
Файл кода с описанием класса VelocityBiotSavart 656
Velocity/VelocityBiotSavart.h
Заголовочный файл с описанием класса VelocityBiotSavart 657
VirtualWake/VirtualWake.cpp
Файл кода с описанием класса VirtualWake 659
VirtualWake/VirtualWake.h
Заголовочный файл с описанием класса VirtualWake 659
VM2D/VM2D.cpp
Основной файл программы VM2D 661
Vortex2D/Vortex2D.cpp
Файл кода с описанием класса Vortex2D 664
Vortex2D/Vortex2D.h
Заголовочный файл с описанием класса Vortex2D 665
Wake/Wake.cpp
Файл кода с описанием класса Wake 667
Wake/Wake.h
Заголовочный файл с описанием класса Wake 667
WakeDataBase/WakeDataBase.cpp
Файл кода с описанием класса WakeDataBase 669
WakeDataBase/WakeDataBase.h
Заголовочный файл с описанием класса WakeDataBase 669
World2D/World2D.cpp
Файл кода с описанием класса World2D 671
World2D/World2D.h
Заголовочный файл с описанием класса World2D 671

Глава 8

Группы

8.1 Параллельные функции

Функции

- virtual void Airfoil::GetDiffVelocityI0I3ToSetOfPointsAndViscousStresses (const WakeDataBase &pointsDb, std::vector< double > &domainRadius, std::vector< double > &I0, std::vector< Point2D > &I3)=0

  Вычисление числителей и знаменателей диффузионных скоростей в заданном наборе точек, обусловленных геометрией профиля, и вычисление вязкого трения
- void Boundary::VirtualWakeSynchronize ()

  MPI-синхронизация виртуального вихревого следа
- virtual void Boundary::GetWakeInfluence (std::vector< double > &wakeVelo) const =0

  Генерация вектора влияния вихревого следа на профиль
- virtual void Boundary::GetConvVelocityToSetOfPoints (const std::vector< Vortex2D > &points, std::vector< Point2D > &velo) const =0

  Вычисление конвективных скоростей в наборе точек, вызываемых наличием завихренности и источников на профиле как от слоев
- virtual void Boundary::GetConvVelocityToSetOfPointsFromVirtualVortexes (const WakeDataBase &pointsDb, std::vector< Point2D > &velo) const =0

  Вычисление конвективных скоростей в наборе точек, вызываемых наличием завихренности и источников на профиле как виртуальных вихрей
- virtual void Velocity::CalcConvVeloToSetOfPoints (const WakeDataBase &pointsDb, std::vector< Point2D > &velo, std::vector< double > &domainRadius, bool onlyRadius=false)=0

  Вычисление конвективных скоростей и радиусов вихревых доменов в заданном наборе точек от следа
- virtual void Velocity::CalcDiffVeloI1I2ToSetOfPoints (const WakeDataBase &pointsDb, const std↩::vector< double > &domainRadius, const WakeDataBase &vorticesDb, std::vector< double > &I1, std::vector< Point2D > &I2)=0

  Вычисление числителей и знаменателей диффузионных скоростей в заданном наборе точек
- void Wake::Inside (const std::vector< Point2D > &newPos, Airfoil &afl, bool isMoves, const Airfoil &oldAfl)

  Проверка пересечения вихрями следа профиля при перемещении
- void WakeDataBase::WakeSynchronize ()

  MPI-синхронизация вихревого следа

8.1.1 Подробное описание

8.1.2 Функции

8.1.2.1 CalcConvVeloToSetOfPoints()

```
virtual void Velocity::CalcConvVeloToSetOfPoints (
            const WakeDataBase & pointsDb,
            std::vector< Point2D > & velo,
            std::vector< double > & domainRadius,
            bool onlyRadius = false )   [pure virtual]
```

#include <Velocity/Velocity.h>

Вычисление конвективных скоростей и радиусов вихревых доменов в заданном наборе точек от следа

Аргументы

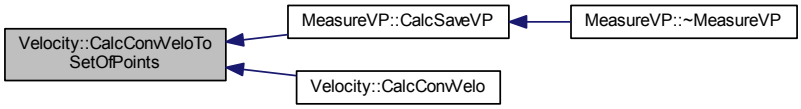
in	pointsDb	константная ссылка на базу данных пелены из вихрей, в которых надо сосчитать конвективные скорости
out	velo	ссылка на вектор скоростей в требуемых точках
out	domainRadius	ссылка на вектор радиусов вихревых доменов
in	onlyRadius	признак вычисления только радиусов доменом, иначе и скоростей тоже (по умолчанию false)

Предупреждения

Использует OMP, MPI

Замещается в VelocityBiotSavart.

Граф вызова функции:



8.1.2.2 CalcDiffVeloI1I2ToSetOfPoints()

```
virtual void Velocity::CalcDiffVeloI1I2ToSetOfPoints (
            const WakeDataBase & pointsDb,
            const std::vector< double > & domainRadius,
            const WakeDataBase & vorticesDb,
            std::vector< double > & I1,
            std::vector< Point2D > & I2 )   [pure virtual]
```

#include <Velocity/Velocity.h>

Вычисление числителей и знаменателей диффузионных скоростей в заданном наборе точек

Аргументы

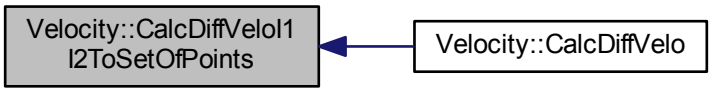
in	pointsDb	константная ссылка на базу данных пелены из вихрей, в которых надо сосчитать диффузионные скорости
in	domainRadius	константная ссылка на вектор радиусов вихревых доменов
in	vorticesDb	константная ссылка на на базу данных пелены из вихрей,от которых надо сосчитать влияния на points
out	I1	ссылка на вектор величин I1 (знаменателей в диффузионных скоростях) в требуемых точках
out	I2	ссылка на вектор величин I2 (числителей в диффузионных скоростях) в требуемых точках

Предупреждения

Использует OMP, MPI

Замещается в VelocityBiotSavart.

Граф вызова функции:



8.1.2.3 GetConvVelocityToSetOfPoints()

```
virtual void Boundary::GetConvVelocityToSetOfPoints (
            const std::vector< Vortex2D > & points,
            std::vector< Point2D > & velo ) const   [pure virtual]
```

#include <Boundary/Boundary.h>

Вычисление конвективных скоростей в наборе точек, вызываемых наличием завихренности и источников на профиле как от слоев

Вычисляет конвективные скорости в наборе точек, которые вызваны влиянием завихренности и источников на профиле как от слоев

Аргументы

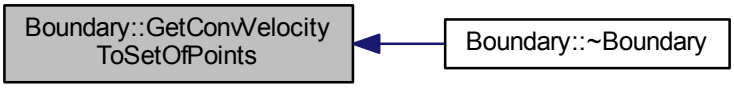
in	points	константная ссылка на набор точек, в которых вычисляются скорости
out	velo	ссылка на вектор скоростей, которые приобретают точки из-за влияния завихренности и источников на профиле

Предупреждения

velo — накапливается!
Использует OMP, MPI

Замещается в BoundaryConstLayerAver.

Граф вызова функции:



8.1.2.4 GetConvVelocityToSetOfPointsFromVirtualVortexes()

```
virtual void Boundary::GetConvVelocityToSetOfPointsFromVirtualVortexes (
            const WakeDataBase & pointsDb,
            std::vector< Point2D > & velo ) const   [pure virtual]
```

#include <Boundary/Boundary.h>

Вычисление конвективных скоростей в наборе точек, вызываемых наличием завихренности и источников на профиле как виртуальных вихрей

Вычисляет конвективные скорости в наборе точек, которые вызваны влиянием завихренности и источников на профиле как виртуальных вихрей

Аргументы

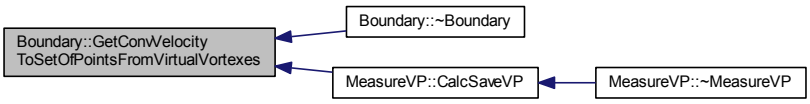
in	pointsDb	константная ссылка на базу данных вихрей, в точках которых вычисляются скорости
out	velo	ссылка на вектор скоростей, которые приобретают точки из-за влияния завихренности и источников на профиле

Предупреждения

velo — накапливается!
Использует OMP, MPI

Замещается в BoundaryConstLayerAver.

Граф вызова функции:



8.1.2.5 GetDiffVelocityI0I3ToSetOfPointsAndViscousStresses()

```
virtual void Airfoil::GetDiffVelocityI0I3ToSetOfPointsAndViscousStresses (
            const WakeDataBase & pointsDb,
            std::vector< double > & domainRadius,
            std::vector< double > & I0,
            std::vector< Point2D > & I3 )   [pure virtual]
```

#include <Airfoil/Airfoil.h>

Вычисление числителей и знаменателей диффузионных скоростей в заданном наборе точек, обусловленных геометрией профиля, и вычисление вязкого трения

Вычисляет диффузионные скорости в наборе точек, которые обусловленных геометрией профиля, и вычисляет вязкое трение

Аргументы

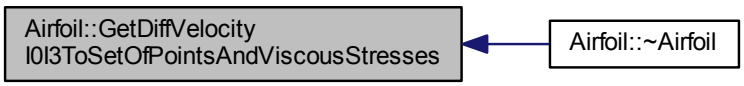
in	pointsDb	константная ссылка на базу данных вихрей, в которых вычисляются скорости
in	domainRadius	ссылка на радиусы вихрей
out	I0	ссылка на вектор знаменателей диффузионных скоростей, которые приобретают точки из-за влияния геометрии профиля
out	I3	ссылка на вектор числителей диффузионных скоростей, которые приобретают точки из-за влияния геометрии профиля

Предупреждения

Векторы I0, I3 — накапливаются!
Использует OMP, MPI

Замещается в AirfoilRect.

Граф вызова функции:



8.1.2.6 GetWakeInfluence()

virtual void Boundary::GetWakeInfluence (
std::vector< double > & wakeVelo) const [pure virtual]

#include <Boundary/Boundary.h>

Генерация вектора влияния вихревого следа на профиль

Генерирует вектор влияния вихревого следа на профиль, используемый затем для расчета вектора правой части.

Аргументы

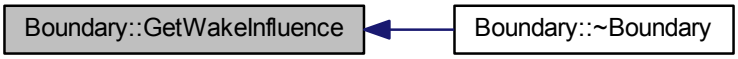
out	wakeVelo	ссылка на вектор влияния вихревого следа на профиль

Предупреждения

Использует OMP, MPI

Замещается в BoundaryConstLayerAver.

Граф вызова функции:



8.1.2.7 Inside()

```
void Wake::Inside (
            const std::vector< Point2D > & newPos,
            Airfoil & afl,
            bool isMoves,
            const Airfoil & oldAfl )
```

#include <Wake/Wake.h>

Проверка пересечения вихрями следа профиля при перемещении

Исполняется сразу для всех вихрей в пелене, осуществляет проверку для отдельного профиля Вихри, попавшие внутрь профиля, получают нулевую циркуляцию, а их "бывшая" циркуляция передается в вектор gammaThrough в структуру данных профиля

Аргументы

in	newPos	константная ссылка на вектор из новых положений вихрей в вихревом следе
in	isMoves	признак того, что профиль подвижный
in	oldAfl	константная ссылка контролируемый профиль до перемещения (используется, если у профиля стоит признак того, что он движется)
in,out	afl	ссылка на контролируемый профиль (происходит изменение afl->gammaThrough)

Предупреждения

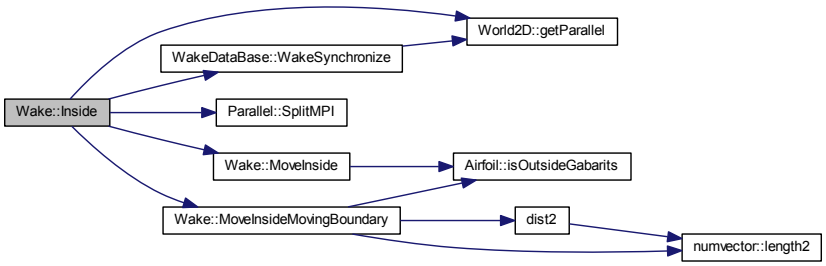
Использует OMP, MPI

См. определение в файле Wake.cpp строка 282

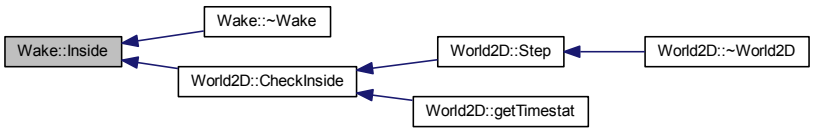
```
283 {
284     int id = W.getParallel().myidWork;
285
286     WakeSynchronize();
287
288     parProp par = W.getParallel().SplitMPI(vtx.size());
289
290     std::vector<Point2D> locNewPos;
291     locNewPos.resize(par.myLen);
292
293     std::vector<double> gamma;
294     gamma.resize(afl.np, 0.0);
295
296     std::vector<double> locGamma;
297     locGamma.resize(afl.np, 0.0);
298
299     std::vector<int> through;
300     if (W.getParallel().myidWork == 0)
301         through.resize(par.totalLen);
302
303     std::vector<int> locThrough;
304     locThrough.resize(par.myLen, -1);
305
306     MPI_Scatterv(const_cast<std::vector<Point2D> &>(newPos).data(), par.len.data(), par.
    disp.data(), Point2D::mpiPoint2D, locNewPos.data(), par.
    myLen, Point2D::mpiPoint2D, 0, W.getParallel().
    commWork);
307
308 #pragma omp parallel for default(none) shared(locGamma, locNewPos, locThrough, afl, oldAfl, par, isMoves)
```

```
    for (int locI = 0; locI < par.myLen; ++locI)
    {
        int i = par.myDisp + locI;
        size_t minN;

        bool crit = isMoves ? MoveInsideMovingBoundary(locNewPos[locI],
vtx[i].r(), oldAfl, afl, minN) : MoveInside(locNewPos[locI], vtx[i].r(), afl, minN);

        if (crit)
            locThrough[static_cast<size_t>(locI)] = static_cast<int>(minN);
    }//for locI

    MPI_Gatherv(locThrough.data(), par.myLen, MPI_INT, through.data(), par.
len.data(), par.disp.data(), MPI_INT, 0, W.getParallel().
commWork);

    //std::ostringstream sss;
    //sss << "through_";
    //std::ofstream throughFile(sss.str());
    //for (size_t i = 0; i < gamma.size(); ++i)
    //  throughFile << gamma[i] << std::endl;
    //throughFile.close();

    if (W.getParallel().myidWork == 0)
    {
        for (size_t q = 0; q < through.size(); ++q)
        if (through[q] > -1)
        {
            locGamma[static_cast<size_t>(through[q])] += vtx[q].g();
            vtx[q].g() = 0.0;
        }
    }

    MPI_Reduce(locGamma.data(), gamma.data(), (int)afl.np, MPI_DOUBLE, MPI_SUM, 0,
W.getParallel().commWork);

    if (W.getParallel().myidWork == 0)
        afl.gammaThrough = gamma;

}//Inside(...)
```

Граф вызовов:



Граф вызова функции:



8.1.2.8 VirtualWakeSynchronize()

void Boundary::VirtualWakeSynchronize ( )

#include <Boundary/Boundary.h>

MPI-синхронизация виртуального вихревого следа

Рассылка следа на все процессоры локальной группы процессоров, занятых решением данной задачи

Предупреждения

Использует OMP, MPI

См. определение в файле Boundary.cpp строка 59

```
60 {
61     int nV;
62     if (W.getParallel().myidWork == 0)
63         nV = static_cast<int>(virtualWake.vtx.size());
64     MPI_Bcast(&nV, 1, MPI_INT, 0, W.getParallel().commWork);
65
66     if (W.getParallel().myidWork > 0)
67         virtualWake.vtx.resize(nV);
68
69     MPI_Bcast(virtualWake.vtx.data(), nV, Vortex2D::mpiVortex2D, 0,
    W.getParallel().commWork);
70 }
```

Граф вызовов:

8.1.2.9 WakeSynchronize()

void WakeDataBase::WakeSynchronize ( )

#include <WakeDataBase/WakeDataBase.h>

MPI-синхронизация вихревого следа

Рассылка следа на все процессоры локальной группы процессоров, занятых решением данной задачи

Предупреждения

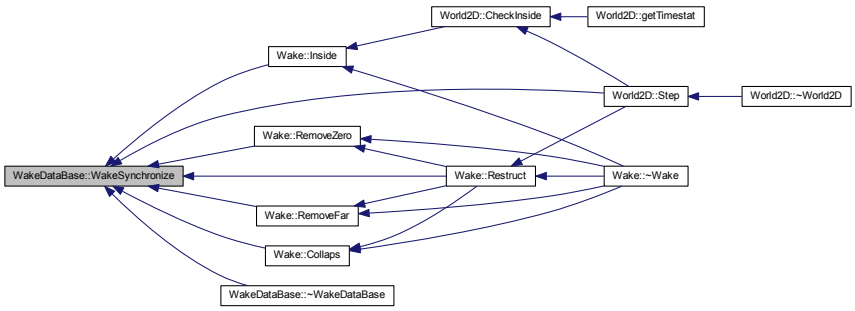
Использует OMP, MPI

См. определение в файле WakeDataBase.cpp строка 71

```
72 {
73     int nV;
74     if (W.getParallel().myidWork == 0)
75         nV = (int)vtx.size();
76     MPI_Bcast(&nV, 1, MPI_INT, 0, W.getParallel().commWork);
77
78     if (W.getParallel().myidWork > 0)
79         vtx.resize(nV);
80
81     MPI_Bcast(vtx.data(), nV, Vortex2D::mpiVortex2D, 0,
    W.getParallel().commWork);
82 }
```

Граф вызовов:

Граф вызова функции:



Глава 9

Пространства имен

9.1 Пространство имен defaults

Глобальные параметры по умолчанию

Функции

- const std::vector< std::string > defaultFileAirfoil ({})

  Список профилей
- const std::string defaultFileWake ("")

  Файл со следом
- const std::string defaultFileSource ("")

  Файл с источниками
- const std::vector< std::string > defaultAirfoil ({})

  Список профилей

Переменные

- const double defaultTimeStart = 0.0

  Начало расчета
- const double defaultTimeAccel = 0.0

  Время разгона
- const int defaultSaveTXT = 1

  Шаг сохранения в текстовый файл
- const int defaultSaveVTK = 0

  Шаг сохранения в бинарный файл
- const int defaultSaveVP = 0

  Шаг подсчета поля скорости и давления
- const int defaultNameLength = 5

  Число разрядов в имени файла
- const double defaultDistFar = 10.0

  Радиус убивания дальнего следа
- const double defaultDelta = 1.e-5

  Расстояние, на которое рождаемый вихрь отодвигается от профиля

- const int defaultVortexPerPanel = 1

  Число вихрей, рождаемых на одной панели
- const double defaultVRef = 0.0

  Референсная скорость, равная нулю, что означает ее равенство скорости набегающего потока
- const std::string defaultAirfoilsDir = "./airfoils/"

  Каталог с файлами профилей
- const std::string defaultWakesDir = "./wakes/"

  Каталог с файлами вихревых следов
- const Point2D defaultBasePoint = { 0.0, 0.0 }

  Базовое смещение профиля
- const double defaultScale = 1.0

  Коэффициент масштабирования профиля
- const double defaultAngle = 0.0

  Угол атаки
- const int defaultPanelsType = 0

  Тип панелей
- const bool defaultInverse = false

  Признак разворота нормалей (для расчета внутреннего течения)
- const int defaultBoundaryCondition = 0

  Способ удовлетворения граничного условия
- const int defaultMechanicalSystemType = 0

  Тип механической системы
- const std::string defaultMechanicalSystem = ""
- const std::string defaultPspFile = "passport"

  Имя файла с паспортом задачи
- const int defaultNp = 1

  Необходимое число процессоров для решения задачи
- const std::string defaultCopyPspFile = ""

  Имя файла с паспортом задачи для копирования в новые каталоги
- static std::ostream * defaultQueueLogStream = &std::cout

  Поток вывода логов и ошибок очереди
- static std::ostream * defaultWorld2DLogStream = &std::cout

  Поток вывода логов и ошибок задачи

9.1.1 Подробное описание

Глобальные параметры по умолчанию

Они могут не указываться ни в параметрах задач, ни в файле defaults

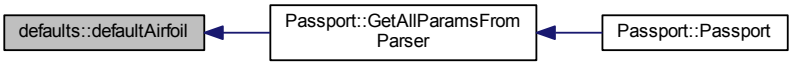
9.1.2 Функции

9.1.2.1 defaultAirfoil()

```
const std::vector<std::string> defaults::defaultAirfoil (
                {} )
```

Список профилей

Граф вызова функции:



9.1.2.2 defaultFileAirfoil()

```
const std::vector<std::string> defaults::defaultFileAirfoil (
                {} )
```

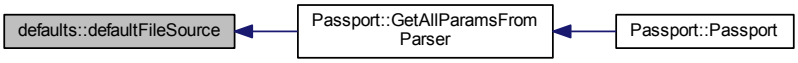
Список профилей

9.1.2.3 defaultFileSource()

```
const std::string defaults::defaultFileSource (
                "" )
```

Файл с источниками

Граф вызова функции:

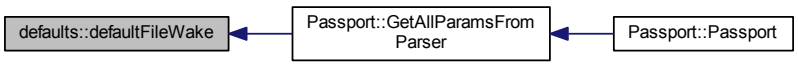


9.1.2.4 defaultFileWake()

```
const std::string defaults::defaultFileWake (
                    "" )
```

Файл со следом

Граф вызова функции:



9.1.3 Переменные

9.1.3.1 defaultAirfoilsDir

```
const std::string defaults::defaultAirfoilsDir = "./airfoils/"
```

Каталог с файлами профилей

См. определение в файле defs.h строка 105

9.1.3.2 defaultAngle

```
const double defaults::defaultAngle = 0.0
```

Угол атаки

См. определение в файле defs.h строка 129

9.1.3.3 defaultBasePoint

```
const Point2D defaults::defaultBasePoint = { 0.0, 0.0 }
```

Базовое смещение профиля

См. определение в файле defs.h строка 123

9.1.3.4 defaultBoundaryCondition

const int defaults::defaultBoundaryCondition = 0

Способ удовлетворения граничного условия

См. определение в файле defs.h строка 138

9.1.3.5 defaultCopyPspFile

const std::string defaults::defaultCopyPspFile = ""

Имя файла с паспортом задачи для копирования в новые каталоги

См. определение в файле defs.h строка 151

9.1.3.6 defaultDelta

const double defaults::defaultDelta = 1.e-5

Расстояние, на которое рождаемый вихрь отодвигается от профиля

См. определение в файле defs.h строка 96

9.1.3.7 defaultDistFar

const double defaults::defaultDistFar = 10.0

Радиус убивания дальнего следа

См. определение в файле defs.h строка 93

9.1.3.8 defaultInverse

const bool defaults::defaultInverse = false

Признак разворота нормалей (для расчета внутреннего течения)

См. определение в файле defs.h строка 135

9.1.3.9 defaultMechanicalSystem

const std::string defaults::defaultMechanicalSystem = ""

См. определение в файле defs.h строка 142

9.1.3.10 defaultMechanicalSystemType

const int defaults::defaultMechanicalSystemType = 0

Тип механической системы

См. определение в файле defs.h строка 141

9.1.3.11 defaultNameLength

const int defaults::defaultNameLength = 5

Число разрядов в имени файла

См. определение в файле defs.h строка 90

9.1.3.12 defaultNp

const int defaults::defaultNp = 1

Необходимое число процессоров для решения задачи

См. определение в файле defs.h строка 148

9.1.3.13 defaultPanelsType

const int defaults::defaultPanelsType = 0

Тип панелей

См. определение в файле defs.h строка 132

9.1.3.14 defaultPspFile

const std::string defaults::defaultPspFile = "passport"

Имя файла с паспортом задачи

См. определение в файле defs.h строка 145

9.1.3.15 defaultQueueLogStream

std::ostream* defaults::defaultQueueLogStream = &std::cout [static]

Поток вывода логов и ошибок очереди

См. определение в файле defs.h строка 154

9.1.3.16 defaultSaveTXT

const int defaults::defaultSaveTXT = 1

Шаг сохранения в текстовый файл

См. определение в файле defs.h строка 81

9.1.3.17 defaultSaveVP

const int defaults::defaultSaveVP = 0

Шаг подсчета поля скорости и давления

См. определение в файле defs.h строка 87

9.1.3.18 defaultSaveVTK

const int defaults::defaultSaveVTK = 0

Шаг сохранения в бинарный файл

См. определение в файле defs.h строка 84

9.1.3.19 defaultScale

const double defaults::defaultScale = 1.0

Коэффициент масштабирования профиля

См. определение в файле defs.h строка 126

9.1.3.20 defaultTimeAccel

const double defaults::defaultTimeAccel = 0.0

Время разгона

См. определение в файле defs.h строка 78

9.1.3.21 defaultTimeStart

const double defaults::defaultTimeStart = 0.0

Начало расчета

См. определение в файле defs.h строка 75

9.1.3.22 defaultVortexPerPanel

const int defaults::defaultVortexPerPanel = 1

Число вихрей, рождаемых на одной панели

См. определение в файле defs.h строка 99

9.1.3.23 defaultVRef

const double defaults::defaultVRef = 0.0

Референсная скорость, равная нулю, что означает ее равенство скорости набегающего потока

См. определение в файле defs.h строка 102

9.1.3.24 defaultWakesDir

const std::string defaults::defaultWakesDir = "./wakes/"

Каталог с файлами вихревых следов

См. определение в файле defs.h строка 108

9.1.3.25 defaultWorld2DLogStream

std::ostream* defaults::defaultWorld2DLogStream = &std::cout [static]

Поток вывода логов и ошибок задачи

См. определение в файле defs.h строка 157

Глава 10

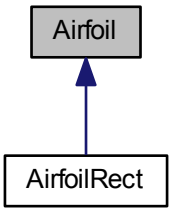
Классы

10.1 Класс Airfoil

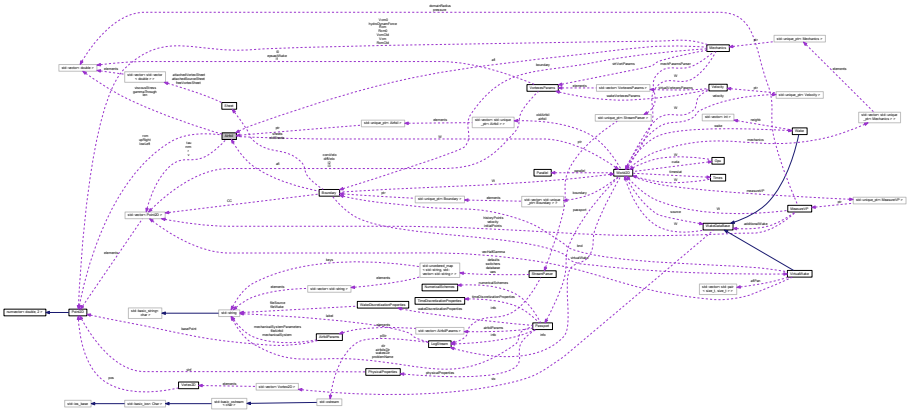
Абстрактный класс, определяющий обтекаемый профиль

#include <Airfoil.h>

Граф наследования:Airfoil:



Граф связей класса Airfoil:



Открытые члены

- IFCUDA (mutable double *devRPtr)

  Указатель на девайсе, где хранятся вершины профиля
- IFCUDA (mutable double *devRhsPtr)

  Указатель на девайсе, где хранится правая часть матрицы
- IFCUDA (mutable std::vector< double > tmpRhs)

  Указатель на хосте, где хранится временная часть матрицы, полученная с девайса
- IFCUDA (mutable double *devFreeVortexSheetPtr)

  Указатель на девайсе, где хранятся интенсивности свободного вихревого слоя на панелях
- IFCUDA (mutable double *devAttachedVortexSheetPtr)

  Указатель на девайсе, где хранятся интенсивности присоединенного вихревого слоя на панелях
- IFCUDA (mutable double *devAttachedSourceSheetPtr)

  Указатель на девайсе, где хранятся интенсивности присоединенного слоя источников на панелях
- Airfoil (const World2D &W_, const size_t numberInPassport_)
- Airfoil (const Airfoil &afl)
- virtual ~Airfoil ()

  Деструктор
- void CalcNrmTauLen ()

  Вычисление нормалей, касательных и длин панелей по текущему положению вершин
- bool isAfter (size_t i, size_t j) const

  Проверка, идет ли вершина i следом за вершиной j.
- void Rotate (double alpha)

  Поворот профиля
- void Scale (double factor)

  Масштабирование профиля
- void Move (const Point2D &dr)

  Перемещение профиля
- void GetGabarits (double gap=0.02)

  Вычисляет габаритный прямоугольник профиля
- bool isInsideGabarits (const Point2D &r) const

  Определяет, находится ли точка с радиус-вектором $\vec{r}$ внутри габаритного прямоугольника профиля
- bool isOutsideGabarits (const Point2D &r) const

  Определяет, находится ли точка с радиус-вектором $\vec{r}$ вне габаритного прямоугольника профиля
- virtual bool IsPointInAirfoil (const Point2D &point) const =0

  Определяет, находится ли точка с радиус-вектором $\vec{r}$ внутри профиля
- virtual void ReadFromFile (const std::string &dir)=0

  Считывание профиля из файла
- virtual void GetDiffVelocityI0I3ToSetOfPointsAndViscousStresses (const WakeDataBase &points↩Db, std::vector< double > &domainRadius, std::vector< double > &I0, std::vector< Point2D > &I3)=0

  Вычисление числителей и знаменателей диффузионных скоростей в заданном наборе точек, обусловленных геометрией профиля, и вычисление вязкого трения

Открытые атрибуты

- const World2D & W

  Константная ссылка на решаемую задачу
- const size_t numberInPassport

  Номер профиля в паспорте
- Point2D rcm

Положение центра масс профиля

- double phiAfl

  Поворот профиля
- double m

  Масса профиля
- double J

  Полярный момент инерции профиля относительно центра масс
- size_t np

  Число панелей на профиле
- std::vector< Point2D > r

  Координаты начал панелей
- bool inverse

  Признак разворота нормалей (для расчета внутренних течений)
- std::vector< Point2D > nrm

  Нормали к панелям профиля
- std::vector< Point2D > tau

  Касательные к панелям профиля
- std::vector< double > len

  Длины панелей профиля
- std::vector< double > viscousStress

  Касательные напряжения на панелях профиля
- Point2D lowLeft

  Левый нижний угол габаритного прямоугольника профиля
- Point2D upRight

  Правый верхний угол габаритного прямоугольника профиля
- std::vector< Point2D > v

  Скорости начал панелей
- std::vector< double > gammaThrough

  Суммарные циркуляции вихрей, пересекших панели профиля на прошлом шаге

10.1.1 Подробное описание

Абстрактный класс, определяющий обтекаемый профиль

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Airfoil.h строка 58

10.1.2 Конструктор(ы)

10.1.2.1 Airfoil() [1/2]

```
Airfoil::Airfoil (
            const World2D & W_,
            const size_t numberInPassport_ )
```

Конструктор

Аргументы

in	W_	константная ссылка на решаемую задачу
in	numberIn↩ Passport_	номер профиля в паспорте задачи

См. определение в файле Airfoil.cpp строка 45

```
46     : np(0), W(W_), numberInPassport(numberInPassport_)
47 { }
```

10.1.2.2 Airfoil() [2/2]

```
Airfoil::Airfoil (
            const Airfoil & afl )   [inline]
```

Конструктор копирования

Необходимо сделать откомментировать — почему нельзя использовать default?

См. определение в файле Airfoil.h строка 149

```
149                         : np(0), W(afl.W), numberInPassport(afl.
    numberInPassport)
150     {
151         rcm = afl.rcm;
152         np = afl.np;
153         phiAfl = afl.phiAfl;
154         inverse = afl.inverse;
155
156         r.insert(r.begin(), afl.r.begin(), afl.r.end());
157         nrm.insert(nrm.begin(), afl.nrm.begin(), afl.nrm.end());
158         tau.insert(tau.begin(), afl.tau.begin(), afl.tau.end());
159         len.insert(len.begin(), afl.len.begin(), afl.len.end());
160         viscousStress = afl.viscousStress;
161         lowLeft = afl.lowLeft;
162         upRight = afl.upRight;
163         v = afl.v;
164         gammaThrough.insert(gammaThrough.begin(), afl.
    gammaThrough.begin(), afl.gammaThrough.end());
165     };
```

10.1.2.3 ~Airfoil()

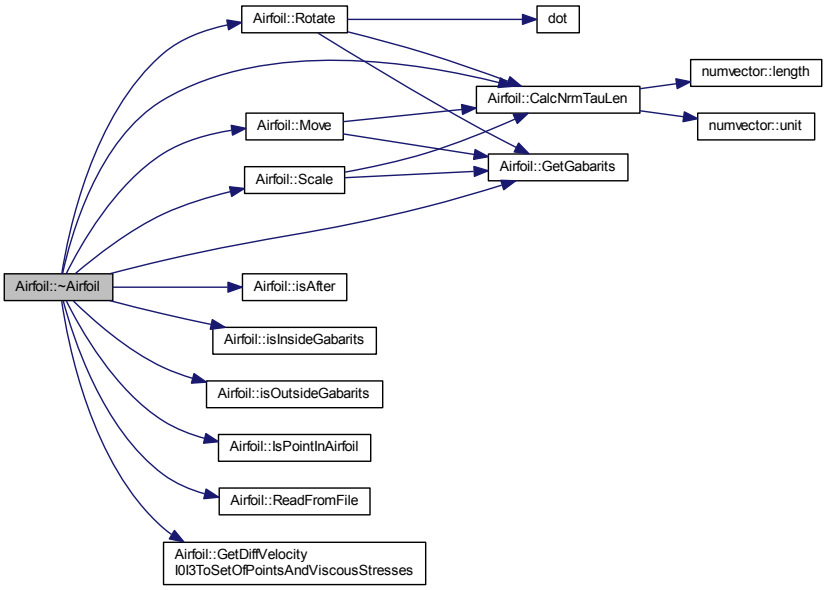
virtual Airfoil::~Airfoil ( ) [inline], [virtual]

Деструктор

См. определение в файле Airfoil.h строка 168

```
168 { };
```

Граф вызовов:



10.1.3 Методы

10.1.3.1 CalcNrmTauLen()

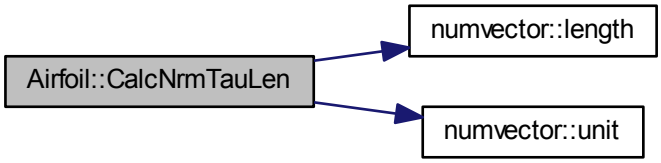
void Airfoil::CalcNrmTauLen ( )

Вычисление нормалей, касательных и длин панелей по текущему положению вершин

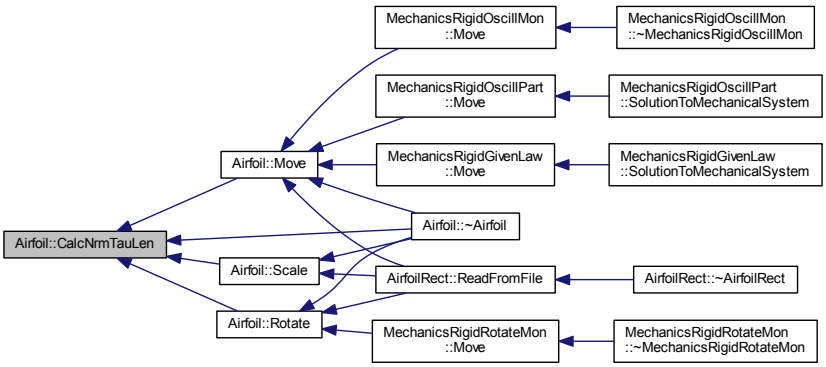
См. определение в файле Airfoil.cpp строка 51

```
{
    if (nrm.size() != np)
    {
        nrm.resize(np);
        tau.resize(np);
        len.resize(np);
    }

    Point2D rpan;

    for (size_t i = 0; i < np; ++i)
    {
        rpan = (r[i + 1] - r[i]);
        len[i] = rpan.length();
        tau[i] = rpan.unit();

        nrm[i] = { tau[i][1], -tau[i][0] };
    }

    //закольцовываем
    nrm.push_back(nrm[0]);
    tau.push_back(tau[0]);
    len.push_back(len[0]);
}//CalcNrmTauLen()
```

Граф вызовов:



Граф вызова функции:



10.1.3.2 GetGabarits()

```
void Airfoil::GetGabarits (
            double gap = 0.02 )
```

Вычисляет габаритный прямоугольник профиля

Заполняет значения полей lowLeft и upRight по габаритному прямоугольнику профиля с учетом зазора

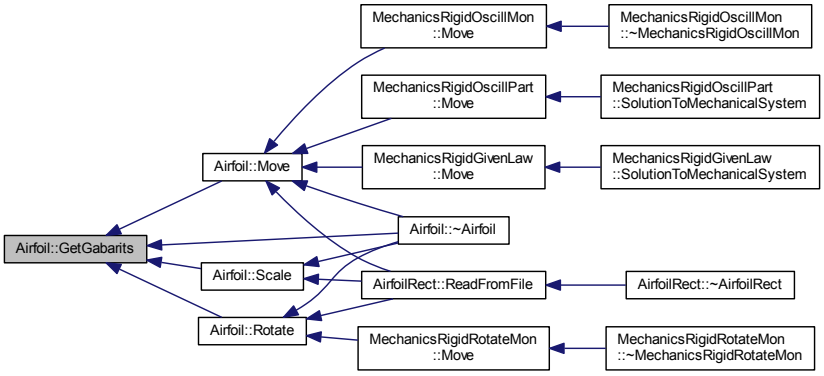
Аргументы

in	gap	относительная величина зазора в долях от размера габаритного прямоугольника (по умолчанию 0.02, что соответствует 2 %)

См. определение в файле Airfoil.cpp строка 122

```
{
    lowLeft = {  1E+10,  1E+10 };
    upRight = { -1E+10, -1E+10 };

    for (size_t i = 0; i < np + 1; i++)
    {
        lowLeft[0] = std::min(lowLeft[0], r[i][0]);
        lowLeft[1] = std::min(lowLeft[1], r[i][1]);

        upRight[0] = std::max(upRight[0], r[i][0]);
        upRight[1] = std::max(upRight[1], r[i][1]);
    }

    Point2D size = upRight - lowLeft;
    lowLeft -= size*gap;
    upRight += size*gap;
}//GetGabarits(...)
```

Граф вызова функции:



10.1.3.3 IFCUDA() [1/6]

```
Airfoil::IFCUDA (
            mutable double * devRPtr )
```

Указатель на девайсе, где хранятся вершины профиля

10.1.3.4 IFCUDA() [2/6]

```
Airfoil::IFCUDA (
            mutable double * devRhsPtr )
```

Указатель на девайсе, где хранится правая часть матрицы

10.1.3.5 IFCUDA() [3/6]

```
Airfoil::IFCUDA (
            mutable std::vector< double > tmpRhs )
```

Указатель на хосте, где хранится временная часть матрицы, полученная с девайса

10.1.3.6 IFCUDA() [4/6]

```
Airfoil::IFCUDA (
            mutable double * devFreeVortexSheetPtr )
```

Указатель на девайсе, где хранятся интенсивности свободного вихревого слоя на панелях

10.1.3.7 IFCUDA() [5/6]

```
Airfoil::IFCUDA (
            mutable double * devAttachedVortexSheetPtr )
```

Указатель на девайсе, где хранятся интенсивности присоединенного вихревого слоя на панелях

10.1.3.8 IFCUDA() [6/6]

```
Airfoil::IFCUDA (
            mutable double * devAttachedSourceSheetPtr )
```

Указатель на девайсе, где хранятся интенсивности присоединенного слоя источников на панелях

10.1.3.9 isAfter()

```
bool Airfoil::isAfter (
            size_t i,
            size_t j ) const
```

Проверка, идет ли вершина i следом за вершиной j.

Аргументы

in	i	проверяемая вершина
in	j	контрольная вершина

Возвращает

true, если i-я вершина следует зп j-й в порядке обхода профиля

См. определение в файле Airfoil.cpp строка 79

```
80 {
81     return ((i == j + 1) || (i == 0 && j == np - 1));
82 }//isAfter(...)
```

Граф вызова функции:

10.1.3.10 isInsideGabarits()

```
bool Airfoil::isInsideGabarits (
                    const Point2D & r ) const
```

Определяет, находится ли точка с радиус-вектором $\vec{r}$ внутри габаритного прямоугольника профиля

Аргументы

in	r	константная ссылка на текущее положение точки

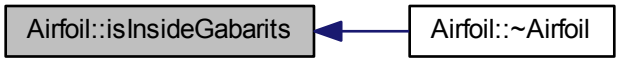
Возвращает

true, если точка внутри габаритного прямоугольника

См. определение в файле Airfoil.cpp строка 143

```
144 {
145     return (r[0] <= upRight[0] && (r[0] >= lowLeft[0] && r[1] >=
    lowLeft[1] && r[1] <= upRight[1]));
146 }//isInsideGabarits(...)
```

Граф вызова функции:



10.1.3.11 isOutsideGabarits()

bool Airfoil::isOutsideGabarits (
const Point2D & r) const

Определяет, находится ли точка с радиус-вектором $\vec{r}$ вне габаритного прямоугольника профиля

Аргументы

in	r	константная ссылка на текущее положение точки

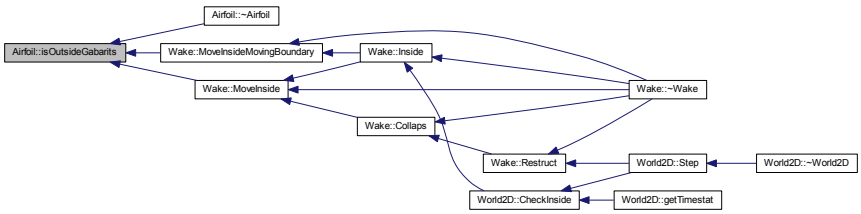
Возвращает

true, если точка вне габаритного прямоугольника

См. определение в файле Airfoil.cpp строка 150

```
151 {
152     return (r[0] > upRight[0] || (r[0] < lowLeft[0] || r[1] <
    lowLeft[1] || r[1] > upRight[1]));
153 }//isOutsideGabarits(...)
```

Граф вызова функции:



10.1.3.12 IsPointInAirfoil()

```
virtual bool Airfoil::IsPointInAirfoil (
            const Point2D & point ) const   [pure virtual]
```

Определяет, находится ли точка с радиус-вектором $\vec{r}$ внутри профиля

Аргументы

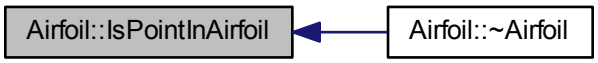
in	point	константная ссылка на текущее положение точки

Возвращает

true, если точка внутри профиля

Замещается в AirfoilRect.

Граф вызова функции:



10.1.3.13 Move()

```
void Airfoil::Move (
            const Point2D & dr )
```

Перемещение профиля

Плоскопараллельно перемещает профиль на вектор $\overrightarrow{dr}$

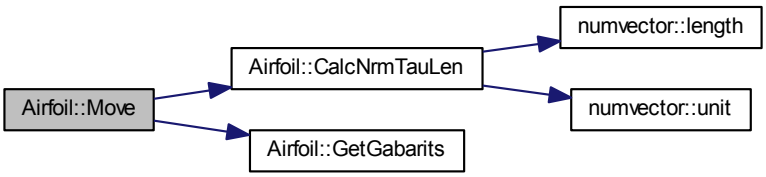
Аргументы

in	dr	константная ссылка на вектор перемещения

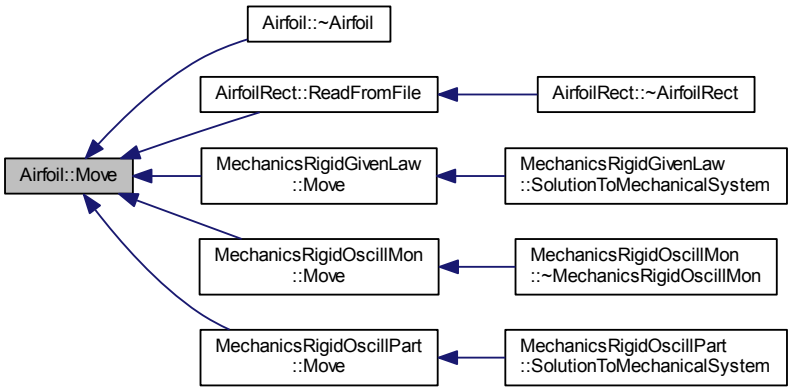
См. определение в файле Airfoil.cpp строка 86

```
{
    for (size_t i = 0; i < np + 1; i++)
        r[i] += dr;
    rcm += dr;
    CalcNrmTauLen();
    GetGabarits();
}//Move(...)
```

Граф вызовов:



Граф вызова функции:



10.1.3.14 ReadFromFile()

virtual void Airfoil::ReadFromFile (
const std::string & dir) [pure virtual]

Считывание профиля из файла

Считывание геометрии профиля из файла, вычисление всех прочих параметров профиля
После загрузки из файла профиль поворачивается на нужный угол и масштабируется на нужный коэффициент

Предупреждения

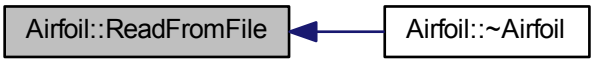
Сейчас масса, момент инерции и скорости вершин зануляются.

Аргументы

in	dir	константная ссылка на строку — имя каталога, где лежит считываемый файл

Замещается в AirfoilRect.

Граф вызова функции:



10.1.3.15 Rotate()

```
void Airfoil::Rotate (
                double alpha )
```

Поворот профиля

Поворачивает профиль на угол α вокруг центра масс

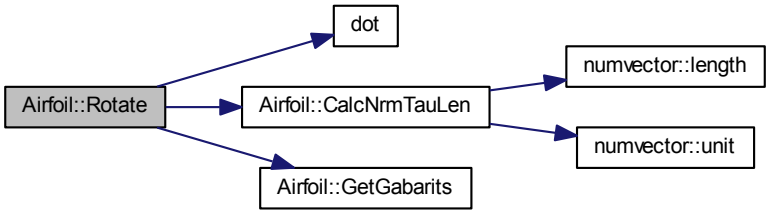
Аргументы

in	alpha	угол поворота против часовой стрелки в радианах

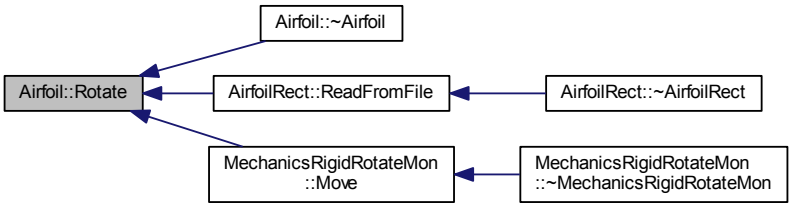
См. определение в файле Airfoil.cpp строка 97

```
{
  phiAfl += alpha;
  numvector<numvector<double, 2>, 2> rotMatrix = { { cos(alpha), sin(alpha)
  }, { -sin(alpha), cos(alpha) } };

  for (size_t i = 0; i < np + 1; i++)
      r[i] = rcm + dot(rotMatrix, r[i] - rcm);

  CalcNrmTauLen();
  GetGabarits();
}//Rotate(...)
```

Граф вызовов:



Граф вызова функции:



10.1.3.16 Scale()

void Airfoil::Scale (
double factor)

Масштабирование профиля

Масштабирует профиль в factor раз относительно центра масс

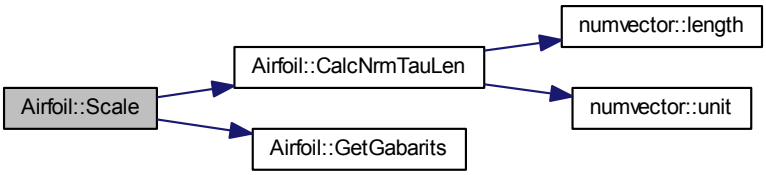
Аргументы

in	factor	масштабный коэффициент

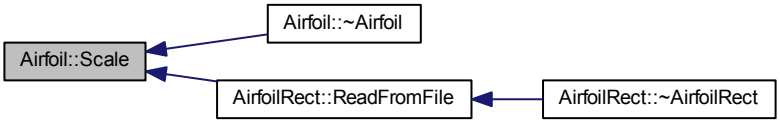
См. определение в файле Airfoil.cpp строка 111

```
112 {
113     for (size_t i = 0; i < np + 1; i++)
114         r[i] = rcm + factor*(r[i] - rcm);
115
116     CalcNrmTauLen(); //строго говоря, меняются при этом только длины, нормали и касательные -
    без изменения
117     GetGabarits();
118 }//Scale(...)
```

Граф вызовов:



Граф вызова функции:



10.1.4 Данные класса

10.1.4.1 gammaThrough

std::vector<double> Airfoil::gammaThrough

Суммарные циркуляции вихрей, пересекших панели профиля на прошлом шаге

Используются в правой части системы, чтобы компенсировать вихри, проникшие в профиль

См. определение в файле Airfoil.h строка 140

10.1.4.2 inverse

bool Airfoil::inverse

Признак разворота нормалей (для расчета внутренних течений)

См. определение в файле Airfoil.h строка 90

10.1.4.3 J

double Airfoil::J

Полярный момент инерции профиля относительно центра масс

См. определение в файле Airfoil.h строка 77

10.1.4.4 len

std::vector<double> Airfoil::len

Длины панелей профиля

См. определение в файле Airfoil.h строка 121

10.1.4.5 lowLeft

Point2D Airfoil::lowLeft

Левый нижний угол габаритного прямоугольника профиля

См. определение в файле Airfoil.h строка 126

10.1.4.6 m

double Airfoil::m

Масса профиля

См. определение в файле Airfoil.h строка 74

10.1.4.7 np

size_t Airfoil::np

Число панелей на профиле

См. определение в файле Airfoil.h строка 80

10.1.4.8 nrm

std::vector<Point2D> Airfoil::nrm

Нормали к панелям профиля

Нормали задаются внешними, нормированными на единицу

См. определение в файле Airfoil.h строка 113

10.1.4.9 numberInPassport

const size_t Airfoil::numberInPassport

Номер профиля в паспорте

См. определение в файле Airfoil.h строка 65

10.1.4.10 phiAfl

double Airfoil::phiAfl

Поворот профиля

См. определение в файле Airfoil.h строка 71

10.1.4.11 r

std::vector<Point2D> Airfoil::r

Координаты начал панелей

Всегда обеспечивается условие r[np] = r[0];
вектор r содержит (np+1) элемент, где np — число панелей.
Это позволяет удобно обращаться к r[i] и r[i+1] как к началу и концу i-й панели

См. определение в файле Airfoil.h строка 87

10.1.4.12 rcm

Point2D Airfoil::rcm

Положение центра масс профиля

См. определение в файле Airfoil.h строка 68

10.1.4.13 tau

std::vector<Point2D> Airfoil::tau

Касательные к панелям профиля

Касательные соответствуют обходу профиля против часовой стрелки, задаются нормированными на единицу

См. определение в файле Airfoil.h строка 118

10.1.4.14 upRight

Point2D Airfoil::upRight

Правый верхний угол габаритного прямоугольника профиля

См. определение в файле Airfoil.h строка 127

10.1.4.15 v

std::vector<Point2D> Airfoil::v

Скорости начал панелей

Всегда обеспечивается условие v[np] = v[0];
вектор v содержит (np+1) элемент, где np — число панелей.
Это позволяет удобно обращаться к v[i] и v[i+1] как к скорости начала и конца i-й панели

Предупреждения

при чтении из файла зашивается нулями

См. определение в файле Airfoil.h строка 135

10.1.4.16 viscousStress

std::vector<double> Airfoil::viscousStress

Касательные напряжения на панелях профиля

См. определение в файле Airfoil.h строка 124

10.1.4.17 W

const World2D& Airfoil::W

Константная ссылка на решаемую задачу

См. определение в файле Airfoil.h строка 62

Объявления и описания членов классов находятся в файлах:

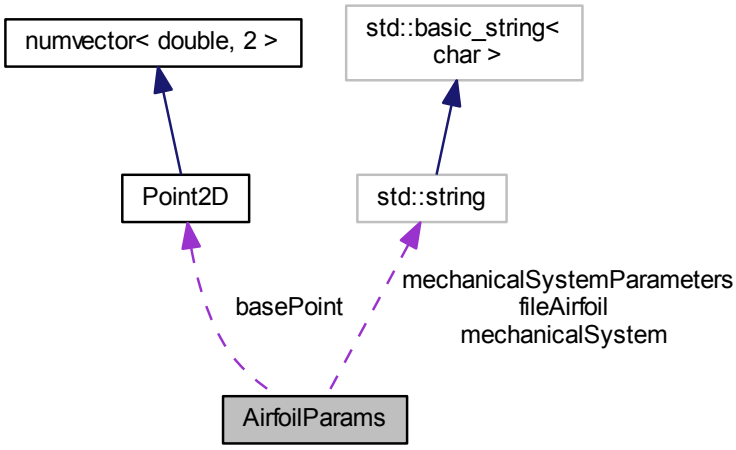
- Airfoil/Airfoil.h
- Airfoil/Airfoil.cpp

10.2 Структура AirfoilParams

Структура, задающая параметры профиля

#include <Passport.h>

Граф связей класса AirfoilParams:



Открытые атрибуты

- std::string fileAirfoil

  Имя файла с начальным состоянием профилей (без полного пути)
- Point2D basePoint

  Смещение центра масс (перенос профиля)
- double scale

  Коэффициент масштабирования
- double angle

Угол поворота (угол атаки)

- bool inverse

  Признак разворота нормалей (для расчета внутреннего течения)
- int panelsType

  Тип панелей (0 — прямые)
- int boundaryCondition

  Метод аппроксимации граничных условий
- int mechanicalSystemType

  Тип механической системы
- std::string mechanicalSystem
- std::string mechanicalSystemParameters

10.2.1 Подробное описание

Структура, задающая параметры профиля

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Passport.h строка 231

10.2.2 Данные класса

10.2.2.1 angle

double AirfoilParams::angle

Угол поворота (угол атаки)

См. определение в файле Passport.h строка 243

10.2.2.2 basePoint

Point2D AirfoilParams::basePoint

Смещение центра масс (перенос профиля)

См. определение в файле Passport.h строка 237

10.2.2.3 boundaryCondition

int AirfoilParams::boundaryCondition

Метод аппроксимации граничных условий

См. определение в файле Passport.h строка 252

10.2.2.4 fileAirfoil

std::string AirfoilParams::fileAirfoil

Имя файла с начальным состоянием профилей (без полного пути)

См. определение в файле Passport.h строка 234

10.2.2.5 inverse

bool AirfoilParams::inverse

Признак разворота нормалей (для расчета внутреннего течения)

См. определение в файле Passport.h строка 246

10.2.2.6 mechanicalSystem

std::string AirfoilParams::mechanicalSystem

См. определение в файле Passport.h строка 256

10.2.2.7 mechanicalSystemParameters

std::string AirfoilParams::mechanicalSystemParameters

См. определение в файле Passport.h строка 257

10.2.2.8 mechanicalSystemType

int AirfoilParams::mechanicalSystemType

Тип механической системы

См. определение в файле Passport.h строка 255

10.2.2.9 panelsType

int AirfoilParams::panelsType

Тип панелей (0 — прямые)

См. определение в файле Passport.h строка 249

10.2.2.10 scale

double AirfoilParams::scale

Коэффициент масштабирования

См. определение в файле Passport.h строка 240

Объявления и описания членов структуры находятся в файле:

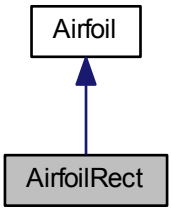
- Passport/Passport.h

10.3 Класс AirfoilRect

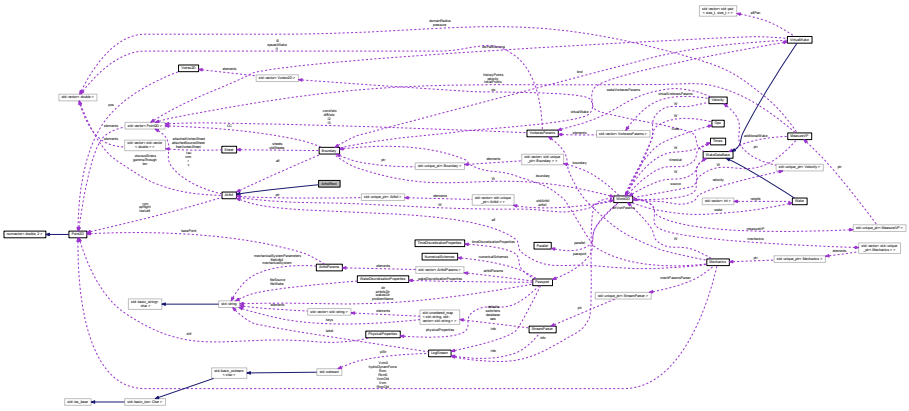
Класс, определяющий тип обтекаемого профиля

#include <AirfoilRect.h>

Граф наследования:AirfoilRect:



Граф связей класса AirfoilRect:



Открытые члены

- AirfoilRect (const World2D &W_, const size_t numberInPassport_)

  Конструктор
- AirfoilRect (const Airfoil &afl)
- virtual ~AirfoilRect ()

  Деструктор
- virtual void ReadFromFile (const std::string &dir)

  Считывание профиля из файла
- virtual bool IsPointInAirfoil (const Point2D &point) const

  Определяет, находится ли точка с радиус-вектором $\vec{r}$ внутри профиля
- virtual void GetDiffVelocityI0I3ToSetOfPointsAndViscousStresses (const WakeDataBase &points↩Db, std::vector< double > &domainRadius, std::vector< double > &I0, std::vector< Point2D > &I3)

Вычисление числителей и знаменателей диффузионных скоростей в заданном наборе точек, обусловленных геометрией профиля, и вычисление вязкого трения

- IFCUDA (mutable double *devRPtr)

  Указатель на девайсе, где хранятся вершины профиля
- IFCUDA (mutable double *devRhsPtr)

  Указатель на девайсе, где хранится правая часть матрицы
- IFCUDA (mutable std::vector< double > tmpRhs)

  Указатель на хосте, где хранится временная часть матрицы, полученная с девайса
- IFCUDA (mutable double *devFreeVortexSheetPtr)

  Указатель на девайсе, где хранятся интенсивности свободного вихревого слоя на панелях
- IFCUDA (mutable double *devAttachedVortexSheetPtr)

  Указатель на девайсе, где хранятся интенсивности присоединенного вихревого слоя на панелях
- IFCUDA (mutable double *devAttachedSourceSheetPtr)

  Указатель на девайсе, где хранятся интенсивности присоединенного слоя источников на панелях
- void CalcNrmTauLen ()

  Вычисление нормалей, касательных и длин панелей по текущему положению вершин
- bool isAfter (size_t i, size_t j) const

  Проверка, идет ли вершина i следом за вершиной j.
- void Rotate (double alpha)

  Поворот профиля
- void Scale (double factor)

  Масштабирование профиля
- void Move (const Point2D &dr)

  Перемещение профиля
- void GetGabarits (double gap=0.02)

  Вычисляет габаритный прямоугольник профиля
- bool isInsideGabarits (const Point2D &r) const

  Определяет, находится ли точка с радиус-вектором $\vec{r}$ внутри габаритного прямоугольника профиля
- bool isOutsideGabarits (const Point2D &r) const

  Определяет, находится ли точка с радиус-вектором $\vec{r}$ вне габаритного прямоугольника профиля

Открытые атрибуты

- const World2D & W

  Константная ссылка на решаемую задачу
- const size_t numberInPassport

  Номер профиля в паспорте
- Point2D rcm

  Положение центра масс профиля
- double phiAfl

  Поворот профиля
- double m

  Масса профиля
- double J

  Полярный момент инерции профиля относительно центра масс
- size_t np

  Число панелей на профиле
- std::vector< Point2D > r

  Координаты начал панелей
- bool inverse

Признак разворота нормалей (для расчета внутренних течений)

- std::vector< Point2D > nrm

  Нормали к панелям профиля
- std::vector< Point2D > tau

  Касательные к панелям профиля
- std::vector< double > len

  Длины панелей профиля
- std::vector< double > viscousStress

  Касательные напряжения на панелях профиля
- Point2D lowLeft

  Левый нижний угол габаритного прямоугольника профиля
- Point2D upRight

  Правый верхний угол габаритного прямоугольника профиля
- std::vector< Point2D > v

  Скорости начал панелей
- std::vector< double > gammaThrough

  Суммарные циркуляции вихрей, пересекших панели профиля на прошлом шаге

10.3.1 Подробное описание

Класс, определяющий тип обтекаемого профиля

Тип профиля:

- профиль с прямолинейными панелями.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле AirfoilRect.h строка 59

10.3.2 Конструктор(ы)

10.3.2.1 AirfoilRect() [1/2]

AirfoilRect::AirfoilRect (
const World2D & W_,
const size_t numberInPassport_) [inline]

Конструктор

См. определение в файле AirfoilRect.h строка 65

```
66        :Airfoil(W_, numberInPassport_)
67    { };
```

10.3.2.2 AirfoilRect() [2/2]

AirfoilRect::AirfoilRect (
const Airfoil & afl) [inline]

См. определение в файле AirfoilRect.h строка 69

```
69 : Airfoil(afl){};
```

10.3.2.3 ~AirfoilRect()

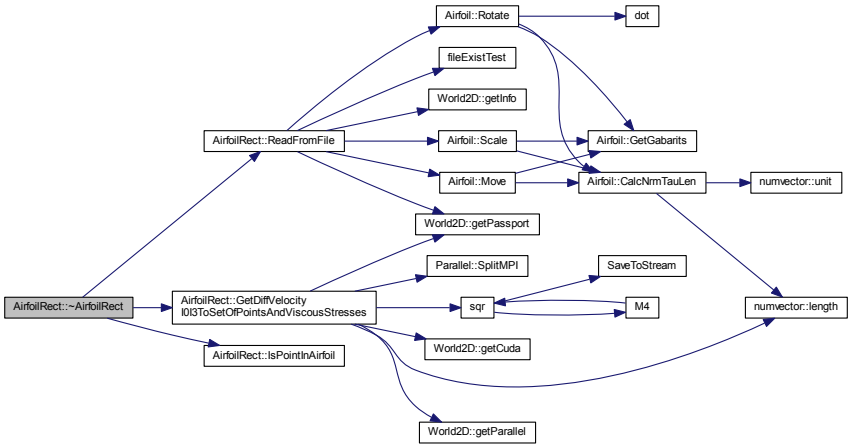
virtual AirfoilRect::~AirfoilRect ( ) [inline], [virtual]

Деструктор

См. определение в файле AirfoilRect.h строка 72

```
72 { };
```

Граф вызовов:



10.3.3 Методы

10.3.3.1 CalcNrmTauLen()

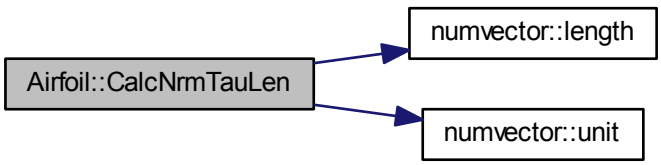
void Airfoil::CalcNrmTauLen ( ) [inherited]

Вычисление нормалей, касательных и длин панелей по текущему положению вершин

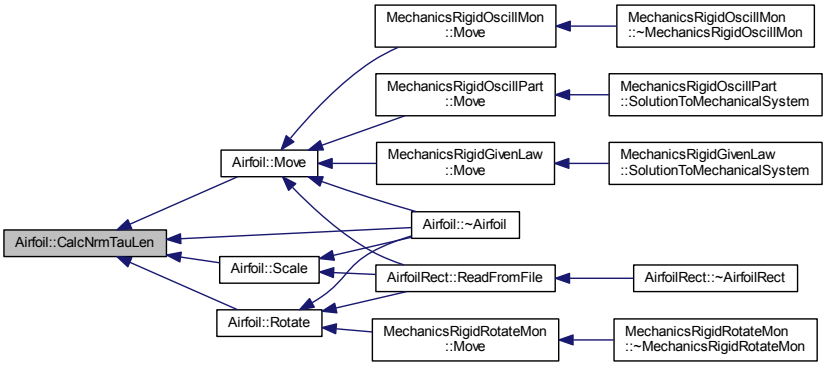
См. определение в файле Airfoil.cpp строка 51

```
52 {
53     if (nrm.size() != np)
54     {
55         nrm.resize(np);
56         tau.resize(np);
57         len.resize(np);
58     }
59
60     Point2D rpan;
61
62     for (size_t i = 0; i < np; ++i)
63     {
64         rpan = (r[i + 1] - r[i]);
65         len[i] = rpan.length();
66         tau[i] = rpan.unit();
67
68         nrm[i] = { tau[i][1], -tau[i][0] };
69     }
70
71     //закольцовываем
72     nrm.push_back(nrm[0]);
73     tau.push_back(tau[0]);
74     len.push_back(len[0]);
75 }//CalcNrmTauLen()
```

Граф вызовов:



Граф вызова функции:



10.3.3.2 GetDiffVelocityI0I3ToSetOfPointsAndViscousStresses()

void AirfoilRect::GetDiffVelocityI0I3ToSetOfPointsAndViscousStresses (
const WakeDataBase & pointsDb,
std::vector< double > & domainRadius,
std::vector< double > & I0,
std::vector< Point2D > & I3) [virtual]

Вычисление числителей и знаменателей диффузионных скоростей в заданном наборе точек, обусловленных геометрией профиля, и вычисление вязкого трения

Вычисляет диффузионные скорости в наборе точек, которые обусловленных геометрией профиля, и вычисляет вязкое трение

Аргументы

in	pointsDb	константная ссылка на базу данных вихрей, в которых вычисляются скорости
in	domainRadius	ссылка на радиусы вихрей
out	I0	ссылка на вектор знаменателей диффузионных скоростей, которые приобретают точки из-за влияния геометрии профиля
out	I3	ссылка на вектор числителей диффузионных скоростей, которые приобретают точки из-за влияния геометрии профиля

Предупреждения

Векторы I0, I3 — накапливаются!
Использует OMP, MPI

Замещает Airfoil.

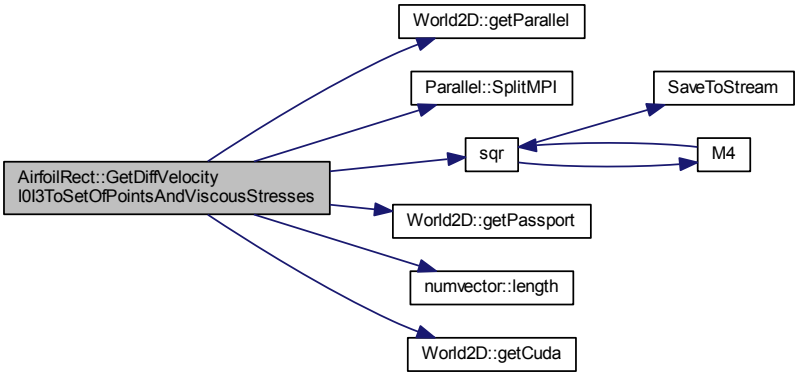
См. определение в файле AirfoilRect.cpp строка 90

```
{
    std::vector<double> selfI0;
    std::vector<Point2D> selfI3;

    viscousStress.clear();
    viscousStress.resize(np, 0.0);

    const int& id = W.getParallel().myidWork;

    parProp par = W.getParallel().SplitMPI(pointsDb.vtx.size());

    std::vector<Vortex2D> locPoints;
    locPoints.resize(par.myLen);

    MPI_Scatterv(const_cast<std::vector<Vortex2D>&>(pointsDb.vtx).data(), par.len.data(), par.disp.data(), Vortex2D::mpiVortex2D, \
        locPoints.data(), par.myLen, Vortex2D::mpiVortex2D, 0, W.getParallel().commWork);

    std::vector<double> locI0(par.myLen, 0.0);
    std::vector<Point2D> locI3(par.myLen, {0.0, 0.0});


#pragma warning (push)
#pragma warning (disable: 4101)
    //Локальные переменные для цикла
    Point2D q, xi, xi_m, v0;
    double lxi, lxi_m, lenj_m;

    Point2D mn;
    int new_n;
    Point2D h;

    Point2D vec;
    double s, d;

    double vs;
    double iDDomRad, expon;
#pragma warning (pop)

    double iDPIepscol2 = 1.0 / (PI * sqr(W.getPassport().wakeDiscretizationProperties.epscol));

#pragma omp parallel for \
    default(none) \
    shared(locI0, locI3, domainRadius, locPoints, id, iDPIepscol2, par) \
    private(xi, xi_m, lxi, lxi_m, lenj_m, v0, q, new_n, mn, h, d, s, vec, vs, expon, iDDomRad) schedule(dynamic, DYN_SCHEDULE)
    for (int i = 0; i < par.myLen; ++i)
    {
        iDDomRad = 1.0 / domainRadius[i+par.myDisp];

        for (size_t j = 0; j < r.size() - 1; j++)
        {
            q = locPoints[i].r() - 0.5 * (r[j] + r[j + 1]);
            vec = tau[j];

            s = q * vec;

            //POLARA
            //Сделала расстояние до центра, а не до конца панели
            /*if (fabs(s) > 0.5 * len[j])
                d = sqrt(sqr(fabs(q*vec) - 0.5*len[j]) + sqr(q*nrm[j]));
            else
                d = fabs(q*nrm[j]);*/
            d = q.length();

            if (d < 50.0 * len[j])    //Почему зависит от длины панели???
            {
                v0 = vec * len[j];

                if (d > 5.0 * len[j])
                {
                    xi = q * iDDomRad;
                    lxi = xi.length();

                    expon = exp(-lxi);
                    mn = len[j] * nrm[j] * expon;

                    if (locI0[i] != -PI * domainRadius[i + par.myDisp])
                    {
                        locI0[i] += xi * mn * (lxi + 1.0) / (lxi*lxi);
                        locI3[i] += mn;
                    }

                    viscousStress[j] += locPoints[i].g() * expon * iDPIepscol2;
```

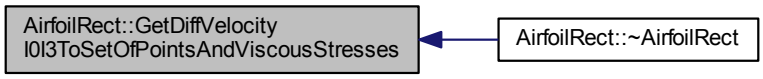
```
            }
            else if ((d <= 5.0 * len[j]) && (d >= 0.1 * len[j]))
            {
                vs = 0.0;
                new_n = static_cast<int>(ceil(5.0 * len[j] / d));
                //new_n = 100;
                h = v0 * (1.0 / new_n);

                for (int m = 0; m < new_n; m++)
                {
                    xi_m = (locPoints[i].r() - (r[j] + h * (m + 0.5))) * iDDomRad;
                    lxi_m = xi_m.length();

                    expon = exp(-lxi_m);
                    lenj_m = len[j] / new_n;
                    mn = lenj_m *nrm[j] * expon;
                    if (locI0[i] != -PI * domainRadius[i + par.myDisp])
                    {
                        locI0[i] += xi_m*  mn * (lxi_m + 1.0) / (lxi_m*lxi_m);
                        locI3[i] += mn;
                    }
                    vs += locPoints[i].g() * expon;
                }//for m
                viscousStress[j] += vs / new_n * iDPIepscol2;
            }
            else if (d <= 0.1 * len[j])
            {
                locI0[i] = -PI * domainRadius[i + par.myDisp];
                //locI3[i] = 2.0 * nrm[j] * domainRadius[i + par.myDisp] * (1.0 - exp(-0.5 * len[j] / domainRadius[i + par.myDisp]));
                //break;

                if (fabs(s) > 0.5 * len[j])
                {
                    locI3[i] = 2.0 * nrm[j] * domainRadius[i + par.myDisp] * (exp(-fabs(s)  * iDDomRad) * sinh(len[j] * iDDomRad / 2.0));
                    //  viscousStress[j] += locPoints[i].g() * (mn.kcross() * tau[j]) / (PI * sqr(epscol));
                    //  viscousStress[j] += locPoints[i].g() * (2.0 * nrm[j] * domainRadius[i+parallel.myDisp] * (1.0 - exp(-0.5 * len[j] / domainRadius[i+parallel.myDisp]))).kcross() * tau[j] / (PI * sqr(epscol) *len[j]);
                    //viscousStress[j] += locPoints[i].g()*(exp(-fabs(s)  * iDDomRad) * sinh(len[j] * iDDomRad / 2.0))  * iDDomRad / len[j]; //locPoints[i].g() *(2.0 *  epscol* (1.0 - exp(-0.5 * len[j] / epscol))) / (PI * sqr(epscol) *len[j]);
                    viscousStress[j] += 2.0 * locPoints[i].g()*(exp(-fabs(s)  * iDDomRad) * sinh(len[j] * iDDomRad / 2.0))  * iDDomRad / (PI * len[j]);
                }
                else
                {
                    locI3[i] = 2.0 * nrm[j] * domainRadius[i + par.myDisp] * (1.0 - exp(-len[j] * iDDomRad / 2.0)*cosh(fabs(s) * iDDomRad));
                    //viscousStress[j] += points[i].g()* (1.0 - exp(-len[j] * iDDomRad / 2.0)*cosh(fabs(s)  * iDDomRad))  * iDDomRad / len[j]; // points[i].g()*(2.0 *  epscol* (1.0 - exp(-0.5 * len[j] / epscol))) / (PI * sqr(epscol) *len[j]);
                    viscousStress[j] += 2.0 * locPoints[i].g()* (1.0 - exp(-len[j] * iDDomRad / 2.0)* cosh(fabs(s)  * iDDomRad))  * iDDomRad / (PI * len[j]);
                }
                break;

            }
        }//if d<50 len
    }//for j
  }//for i

  if (id == 0)
  {
      selfI0.resize(pointsDb.vtx.size(), 0.0);
      selfI3.resize(pointsDb.vtx.size(), {0.0, 0.0});
  }

  MPI_Gatherv(locI0.data(), par.myLen, MPI_DOUBLE,          selfI0.data(), par.len.data(), par.disp.data(), MPI_DOUBLE,          0, W.getParallel().commWork);
  MPI_Gatherv(locI3.data(), par.myLen, Point2D::mpiPoint2D, selfI3.data(), par.len.data(), par.disp.data(), Point2D::mpiPoint2D, 0, W.getParallel().commWork);

  if (id == 0)
  for (size_t i = 0; i < I0.size(); ++i)
  {
      if (I0[i] != -PI * domainRadius[i])
      {
          if (selfI0[i] == -PI * domainRadius[i])
          {
              I0[i] = selfI0[i];
              I3[i] = selfI3[i];
          }
```

```
            else
            {
                I0[i] += selfI0[i];
                I3[i] += selfI3[i];
            }
        }
    }
}; //GetDiffVelocityI0I3ToSetOfPointsAndViscousStresses(...)
```

Граф вызовов:



Граф вызова функции:



10.3.3.3 GetGabarits()

void Airfoil::GetGabarits (
double gap = 0.02) [inherited]

Вычисляет габаритный прямоугольник профиля

Заполняет значения полей lowLeft и upRight по габаритному прямоугольнику профиля с учетом зазора

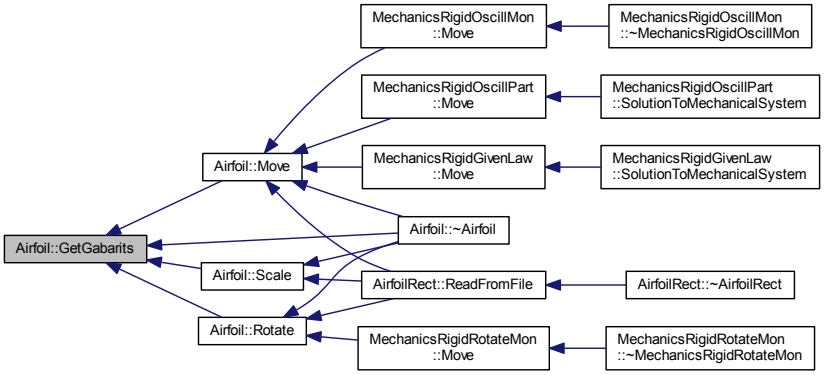
Аргументы

in	gap	относительная величина зазора в долях от размера габаритного прямоугольника (по умолчанию 0.02, что соответствует 2 %)

См. определение в файле Airfoil.cpp строка 122

```
123 {
124     lowLeft = {  1E+10,  1E+10 };
125     upRight = { -1E+10, -1E+10 };
126
127     for (size_t i = 0; i < np + 1; i++)
128     {
129         lowLeft[0] = std::min(lowLeft[0], r[i][0]);
130         lowLeft[1] = std::min(lowLeft[1], r[i][1]);
131
132         upRight[0] = std::max(upRight[0], r[i][0]);
133         upRight[1] = std::max(upRight[1], r[i][1]);
134     }
135
136     Point2D size = upRight - lowLeft;
137     lowLeft -= size*gap;
138     upRight += size*gap;
139 }//GetGabarits(...)
```

Граф вызова функции:



10.3.3.4 IFCUDA() [1/6]

Airfoil::IFCUDA (
mutable double * devRPtr) [inherited]

Указатель на девайсе, где хранятся вершины профиля

10.3.3.5 IFCUDA() [2/6]

Airfoil::IFCUDA (
mutable double * devRhsPtr) [inherited]

Указатель на девайсе, где хранится правая часть матрицы

10.3.3.6 IFCUDA() [3/6]

Airfoil::IFCUDA (
mutable std::vector< double > tmpRhs) [inherited]

Указатель на хосте, где хранится временная часть матрицы, полученная с девайса

10.3.3.7 IFCUDA() [4/6]

Airfoil::IFCUDA (
mutable double * devFreeVortexSheetPtr) [inherited]

Указатель на девайсе, где хранятся интенсивности свободного вихревого слоя на панелях

10.3.3.8 IFCUDA() [5/6]

Airfoil::IFCUDA (
mutable double * devAttachedVortexSheetPtr) [inherited]

Указатель на девайсе, где хранятся интенсивности присоединенного вихревого слоя на панелях

10.3.3.9 IFCUDA() [6/6]

Airfoil::IFCUDA (
mutable double * devAttachedSourceSheetPtr) [inherited]

Указатель на девайсе, где хранятся интенсивности присоединенного слоя источников на панелях

10.3.3.10 isAfter()

bool Airfoil::isAfter (
size_t i,
size_t j) const [inherited]

Проверка, идет ли вершина i следом за вершиной j.

Аргументы

in	i	проверяемая вершина
in	j	контрольная вершина

Возвращает

true, если i-я вершина следует зп j-й в порядке обхода профиля

См. определение в файле Airfoil.cpp строка 79

```
80 {
81     return ((i == j + 1) || (i == 0 && j == np - 1));
82 }//isAfter(...)
```

Граф вызова функции:

10.3.3.11 isInsideGabarits()

```
bool Airfoil::isInsideGabarits (
            const Point2D & r ) const   [inherited]
```

Определяет, находится ли точка с радиус-вектором $\vec{r}$ внутри габаритного прямоугольника профиля

Аргументы

in	r	константная ссылка на текущее положение точки

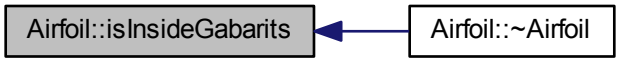
Возвращает

true, если точка внутри габаритного прямоугольника

См. определение в файле Airfoil.cpp строка 143

```
144 {
145     return (r[0] <= upRight[0] && (r[0] >= lowLeft[0] && r[1] >=
    lowLeft[1] && r[1] <= upRight[1]));
146 }//isInsideGabarits(...)
```

Граф вызова функции:



10.3.3.12 isOutsideGabarits()

bool Airfoil::isOutsideGabarits (
const Point2D & r) const [inherited]

Определяет, находится ли точка с радиус-вектором $\vec{r}$ вне габаритного прямоугольника профиля

Аргументы

in	r	константная ссылка на текущее положение точки

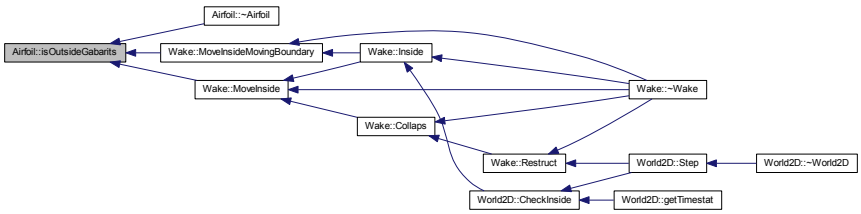
Возвращает

true, если точка вне габаритного прямоугольника

См. определение в файле Airfoil.cpp строка 150

```
151 {
152     return (r[0] > upRight[0] || (r[0] < lowLeft[0] || r[1] <
    lowLeft[1] || r[1] > upRight[1]));
153 }//isOutsideGabarits(...)
```

Граф вызова функции:



10.3.3.13 IsPointInAirfoil()

```
bool AirfoilRect::IsPointInAirfoil (
            const Point2D & point ) const   [virtual]
```

Определяет, находится ли точка с радиус-вектором $\vec{r}$ внутри профиля

Аргументы

in	point	константная ссылка на текущее положение точки

Возвращает

true, если точка внутри профиля

Замещает Airfoil.

См. определение в файле AirfoilRect.cpp строка 308

```
309 {
310     double angle = 0.0;
311
312     Point2D v1, v2;
313
314     for (size_t i = 0; i < r.size() - 1; i++)
315     {
316         v1 = r[i] - point;
317         v2 = r[i+1] - point;
318         angle += atan2(v1^v2, v1 * v2);
319     }
320
321     if (fabs(angle) < 0.1)
322         return false;
323     else return true;
324 }
```

Граф вызова функции:

10.3.3.14 Move()

void Airfoil::Move (
const Point2D & dr) [inherited]

Перемещение профиля

Плоскопараллельно перемещает профиль на вектор $\overrightarrow{dr}$

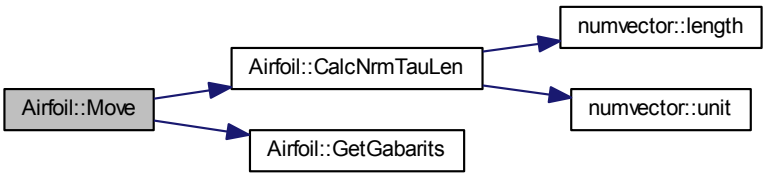
Аргументы

in	dr	константная ссылка на вектор перемещения

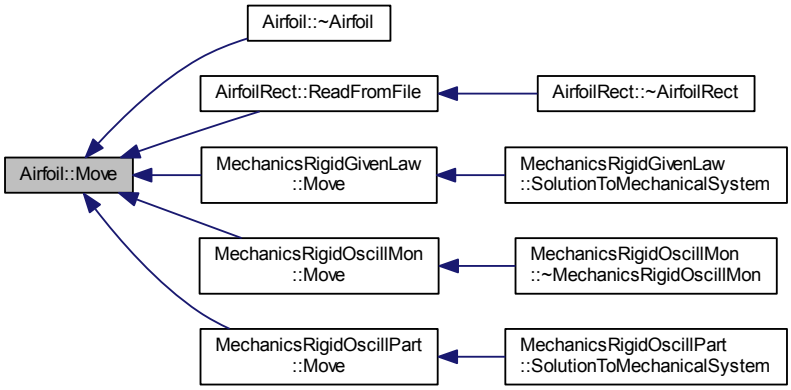
См. определение в файле Airfoil.cpp строка 86

```
87 {
88     for (size_t i = 0; i < np + 1; i++)
89         r[i] += dr;
90     rcm += dr;
91     CalcNrmTauLen();
92     GetGabarits();
93 }//Move(...)
```

Граф вызовов:



Граф вызова функции:



10.3.3.15 ReadFromFile()

void AirfoilRect::ReadFromFile (
const std::string & dir) [virtual]

Считывание профиля из файла

Считывание геометрии профиля из файла, вычисление всех прочих параметров профиля
После загрузки из файла профиль поворачивается на нужный угол и масштабируется на нужный коэффициент

Предупреждения

Сейчас масса, момент инерции и скорости вершин зануляются.

Аргументы

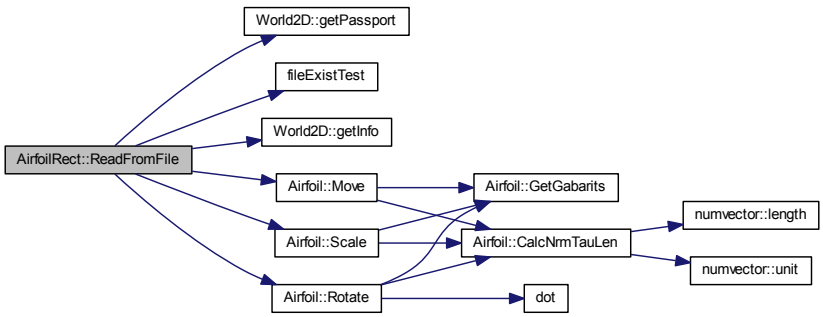
in	dir	константная ссылка на строку — имя каталога, где лежит считываемый файл

Замещает Airfoil.

См. определение в файле AirfoilRect.cpp строка 45

```
46 {
47     const AirfoilParams& param = W.getPassport().
   airfoilParams[numberInPassport];
48     std::string filename = dir + param.fileAirfoil;
49     std::ifstream airfoilFile;
50
51     if (fileExistTest(filename, W.getInfo()))
52     {
53         std::stringstream airfoilFile(Preprocessor(filename).resultString);
54
55         StreamParser airfoilParser(W.getInfo(), "airfoil file parser", airfoilFile);
56
57         m = 0.0; //TODO
58         J = 0.0; //TODO
59         rcm = { 0.0, 0.0 }; //TODO
60         inverse = param.inverse;
61
62
63
64         airfoilParser.get("r", r);
65         if (inverse)
66             std::reverse(r.begin(), r.end());
67
68         //не считываем явно из файла
69         //airfoilParser.get("np", np);
70         np = r.size();
71
72         //замыкаем --- в конец приписываем первую точку (для простоты)
73         r.push_back(r[0]);
74
75         v.resize(0);
76         for (size_t q = 0; q < r.size(); ++q)
77             v.push_back({ 0.0, 0.0 });
78
79         Move(param.basePoint);
80         Scale(param.scale);
81         Rotate(param.angle);
82         //в конце Rotate нормали, касательные и длины вычисляются сами
83
84         gammaThrough.clear();
85         gammaThrough.resize(np, 0.0);
86     }
87 }//ReadFromFile(...)
```

Граф вызовов:



Граф вызова функции:

10.3.3.16 Rotate()

void Airfoil::Rotate (
double alpha) [inherited]

Поворот профиля

Поворачивает профиль на угол α вокруг центра масс

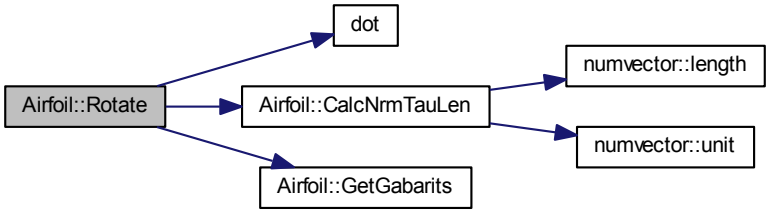
Аргументы

in	alpha	угол поворота против часовой стрелки в радианах

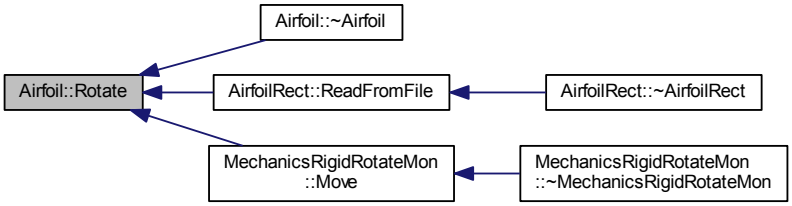
См. определение в файле Airfoil.cpp строка 97

```
98 {
99     phiAfl += alpha;
100     numvector<numvector<double, 2>, 2> rotMatrix = { { cos(alpha), sin(alpha)
      }, { -sin(alpha), cos(alpha) } };
101
102     for (size_t i = 0; i < np + 1; i++)
103         r[i] = rcm + dot(rotMatrix, r[i] - rcm);
104
105     CalcNrmTauLen();
106     GetGabarits();
107 }//Rotate(...)
```

Граф вызовов:



Граф вызова функции:



10.3.3.17 Scale()

void Airfoil::Scale (
double factor) [inherited]

Масштабирование профиля

Масштабирует профиль в factor раз относительно центра масс

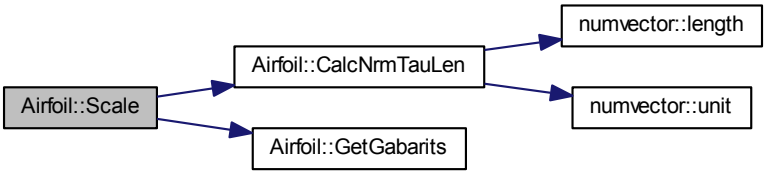
Аргументы

in	factor	масштабный коэффициент

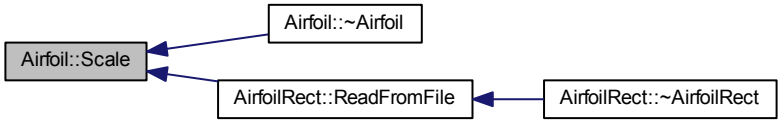
См. определение в файле Airfoil.cpp строка 111

```
{
    for (size_t i = 0; i < np + 1; i++)
        r[i] = rcm + factor*(r[i] - rcm);

    CalcNrmTauLen(); //строго говоря, меняются при этом только длины, нормали и касательные - без изменения
    GetGabarits();
}//Scale(...)
```

Граф вызовов:



Граф вызова функции:



10.3.4 Данные класса

10.3.4.1 gammaThrough

std::vector<double> Airfoil::gammaThrough [inherited]

Суммарные циркуляции вихрей, пересекших панели профиля на прошлом шаге

Используются в правой части системы, чтобы компенсировать вихри, проникшие в профиль

См. определение в файле Airfoil.h строка 140

10.3.4.2 inverse

bool Airfoil::inverse [inherited]

Признак разворота нормалей (для расчета внутренних течений)

См. определение в файле Airfoil.h строка 90

10.3.4.3 J

double Airfoil::J [inherited]

Полярный момент инерции профиля относительно центра масс

См. определение в файле Airfoil.h строка 77

10.3.4.4 len

std::vector<double> Airfoil::len [inherited]

Длины панелей профиля

См. определение в файле Airfoil.h строка 121

10.3.4.5 lowLeft

Point2D Airfoil::lowLeft [inherited]

Левый нижний угол габаритного прямоугольника профиля

См. определение в файле Airfoil.h строка 126

10.3.4.6 m

double Airfoil::m [inherited]

Масса профиля

См. определение в файле Airfoil.h строка 74

10.3.4.7 np

size_t Airfoil::np [inherited]

Число панелей на профиле

См. определение в файле Airfoil.h строка 80

10.3.4.8 nrm

std::vector<Point2D> Airfoil::nrm [inherited]

Нормали к панелям профиля

Нормали задаются внешними, нормированными на единицу

См. определение в файле Airfoil.h строка 113

10.3.4.9 numberInPassport

const size_t Airfoil::numberInPassport [inherited]

Номер профиля в паспорте

См. определение в файле Airfoil.h строка 65

10.3.4.10 phiAfl

double Airfoil::phiAfl [inherited]

Поворот профиля

См. определение в файле Airfoil.h строка 71

10.3.4.11 r

std::vector<Point2D> Airfoil::r [inherited]

Координаты начал панелей

Всегда обеспечивается условие r[np] = r[0];
вектор r содержит (np+1) элемент, где np — число панелей.
Это позволяет удобно обращаться к r[i] и r[i+1] как к началу и концу i-й панели

См. определение в файле Airfoil.h строка 87

10.3.4.12 rcm

Point2D Airfoil::rcm [inherited]

Положение центра масс профиля

См. определение в файле Airfoil.h строка 68

10.3.4.13 tau

std::vector<Point2D> Airfoil::tau [inherited]

Касательные к панелям профиля

Касательные соответствуют обходу профиля против часовой стрелки, задаются нормированными на единицу

См. определение в файле Airfoil.h строка 118

10.3.4.14 upRight

Point2D Airfoil::upRight [inherited]

Правый верхний угол габаритного прямоугольника профиля

См. определение в файле Airfoil.h строка 127

10.3.4.15 v

std::vector<Point2D> Airfoil::v [inherited]

Скорости начал панелей

Всегда обеспечивается условие v[np] = v[0];
вектор v содержит (np+1) элемент, где np — число панелей.
Это позволяет удобно обращаться к v[i] и v[i+1] как к скорости начала и конца i-й панели

Предупреждения

при чтении из файла зашивается нулями

См. определение в файле Airfoil.h строка 135

10.3.4.16 viscousStress

std::vector<double> Airfoil::viscousStress [inherited]

Касательные напряжения на панелях профиля

См. определение в файле Airfoil.h строка 124

10.3.4.17 W

const World2D& Airfoil::W [inherited]

Константная ссылка на решаемую задачу

См. определение в файле Airfoil.h строка 62

Объявления и описания членов классов находятся в файлах:

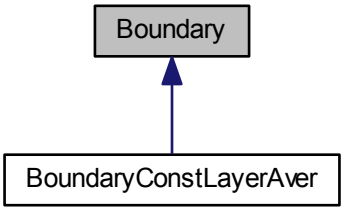
- Airfoil/AirfoilRect.h
- Airfoil/AirfoilRect.cpp

10.4 Класс Boundary

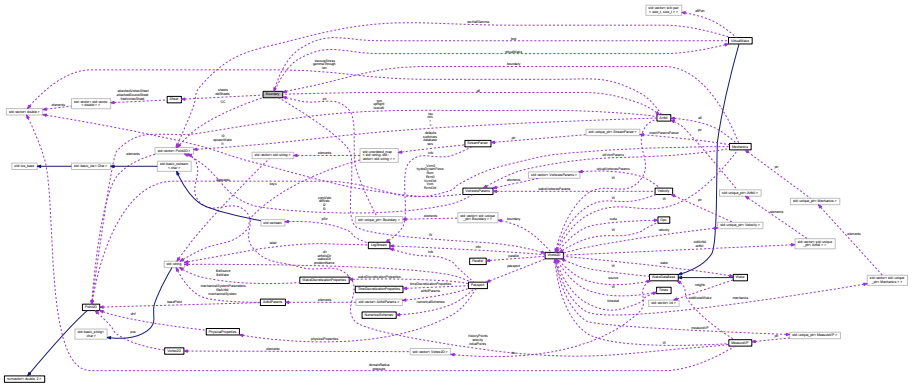
Абстрактный класс, определяющий способ удовлетворения граничного условия на обтекаемом профиле

#include <Boundary.h>

Граф наследования:Boundary:



Граф связей класса Boundary:



Открытые члены

- void VirtualWakeSynchronize ()

  MPI-синхронизация виртуального вихревого следа
- Boundary (const World2D &W_, size_t numberInPassport_, int sheetDim_)

  Конструктор
- virtual ~Boundary ()

  Деструктор
- virtual void FillMatrixSelf (Eigen::MatrixXd &matr, Eigen::VectorXd &lastLine, Eigen::VectorXd &lactCol)=0

  Генерация блока матрицы
- virtual void FillMatrixFromOther (const Boundary &otherBoundary, Eigen::MatrixXd &matr)=0

  Генерация блока матрицы влияния от другого профиля того же типа

- virtual void GetWakeInfluence (std::vector< double > &wakeVelo) const =0

  Генерация вектора влияния вихревого следа на профиль
- virtual void GetConvVelocityToSetOfPoints (const std::vector< Vortex2D > &points, std::vector< Point2D > &velo) const =0

  Вычисление конвективных скоростей в наборе точек, вызываемых наличием завихренности и источников на профиле как от слоев
- virtual void GetConvVelocityToSetOfPointsFromVirtualVortexes (const WakeDataBase &points↩Db, std::vector< Point2D > &velo) const =0

  Вычисление конвективных скоростей в наборе точек, вызываемых наличием завихренности и источников на профиле как виртуальных вихрей
- virtual void GetConvVelocityAtVirtualVortexes (std::vector< Point2D > &velo) const =0

  Вычисление конвективной скорости в точках расположения виртуальных вихрей
- virtual void FillRhs (const Point2D &V0, Eigen::VectorXd &rhs, double *lastRhs, bool move, bool deform)=0

  Заполнение в правой части влияния набегающего потока, следа и присоединенных слоев, действующих от самого себя
- virtual void FillRhsFromOther (const Airfoil &otherAirfoil, Eigen::VectorXd &rhs)=0

  Заполнение в правой части влияния присоединенных слоев, действующих на один профиль от другого
- virtual size_t GetUnknownsSize () const =0

  Возврат размерности вектора решения
- virtual void SolutionToFreeVortexSheetAndVirtualVortex (const Eigen::VectorXd &sol)=0

  Пересчет решения на интенсивность вихревого слоя и на рождаемые вихри на конкретном профиле
- virtual void ComputeAttachedSheetsIntensity ()=0

  Вычисление интенсивностей присоединенного вихревого слоя и присоединенного слоя источников

Открытые атрибуты

- const Airfoil & afl
- const std::vector< Point2D > & CC

  Константная ссылка на вектор из начал (и концов) панелей
- int vortexPerPanel

  Число вихрей, рождаемых на каждой панели профиля и формирующих виртуальный вихревой след
- VirtualWake virtualWake

  Виртуальный вихревой след конкретного профиля
- size_t sheetDim

  Размерность параметров каждого из слоев на каждой из панелей
- Sheet sheets

  Слои на профиле
- Sheet oldSheets

  Слои на профиле с предыдущего шага

Защищенные данные

- const World2D & W

  Константная ссылка на решаемую задачу
- const size_t numberInPassport

  Номер профиля в паспорте

10.4.1 Подробное описание

Абстрактный класс, определяющий способ удовлетворения граничного условия на обтекаемом профиле

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Boundary.h строка 62

10.4.2 Конструктор(ы)

10.4.2.1 Boundary()

```
Boundary::Boundary (
            const World2D & W_,
            size_t numberInPassport_,
            int sheetDim_ )
```

Конструктор

Аргументы

in	W_	константная ссылка на решаемую задачу
in	numberIn↩ Passport_	номер профиля в паспорте задачи
in	sheetDim_	размерность параметра слоя на каждой панели профиля (сколько чисел используется, чтобы описать слой на каждой панели);

См. определение в файле Boundary.cpp строка 43

```
43                                                  :
44    W(W_ ),
45    numberInPassport(numberInPassport_),
46    sheetDim(sheetDim_),
47    afl(W_.getAirfoil(numberInPassport_)),
```

```
    CC(W_.getAirfoil(numberInPassport_).r),
    virtualWake(W_, *this)
{
    sheets.SetLayersDim(afl.np, sheetDim);
    oldSheets.SetLayersDim(afl.np, sheetDim);
    virtualWake.vtx.resize(0);
    virtualWake.vecHalfGamma.resize(0);
}//Boundary(...)
```

Граф вызовов:

10.4.2.2 ~Boundary()

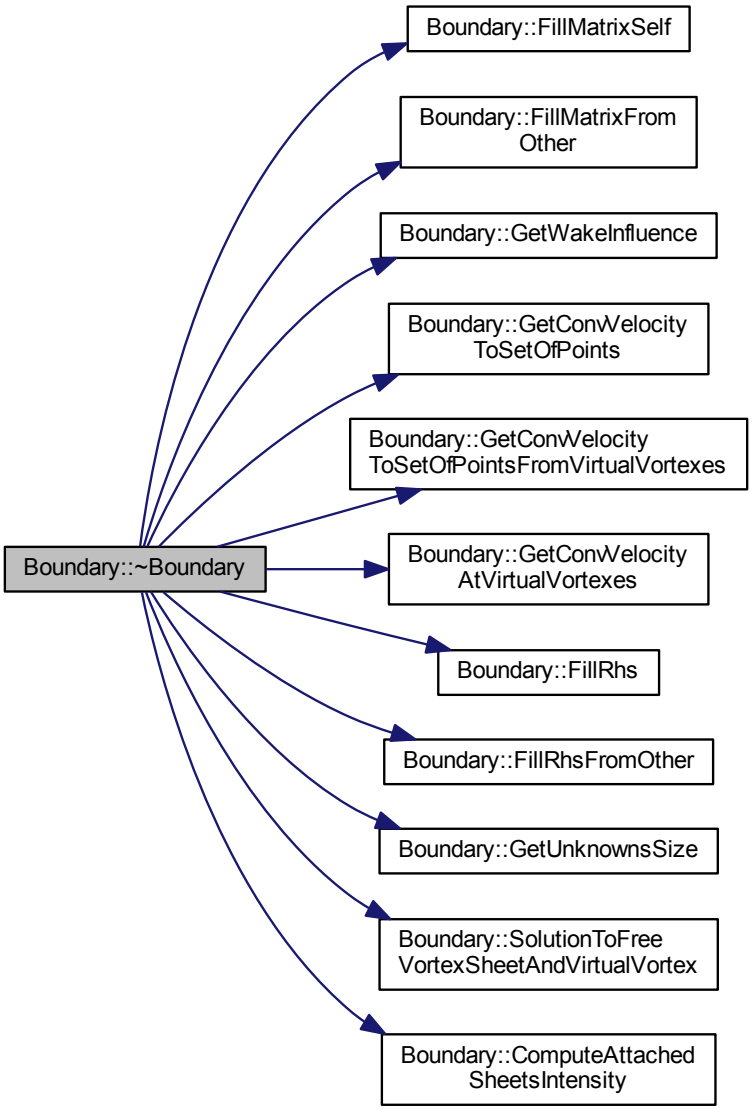
virtual Boundary::~Boundary ( ) [inline], [virtual]

Деструктор

См. определение в файле Boundary.h строка 114

```
114 { };
```

Граф вызовов:



10.4.3 Методы

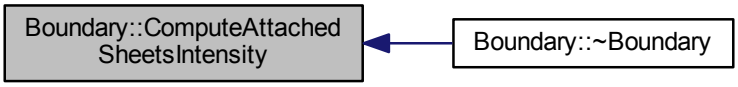
10.4.3.1 ComputeAttachedSheetsIntensity()

virtual void Boundary::ComputeAttachedSheetsIntensity ( ) [pure virtual]

Вычисление интенсивностей присоединенного вихревого слоя и присоединенного слоя источников

Замещается в BoundaryConstLayerAver.

Граф вызова функции:



10.4.3.2 FillMatrixFromOther()

virtual void Boundary::FillMatrixFromOther (
const Boundary & otherBoundary,
Eigen::MatrixXd & matr) [pure virtual]

Генерация блока матрицы влияния от другого профиля того же типа

Генерирует блок матрицы влияния от другого профиля того же типа

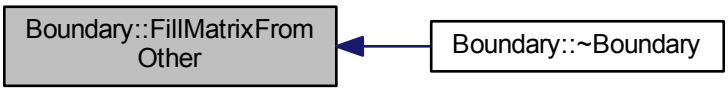
Аргументы

in	otherBoundary	константная ссылка на граничное условие на втором профиле
out	matr	ссылка на генерируемый блок матрицы

Необходимо сделать Пока считается, что граничные условия одинаковые

Замещается в BoundaryConstLayerAver.

Граф вызова функции:



10.4.3.3 FillMatrixSelf()

```
virtual void Boundary::FillMatrixSelf (
            Eigen::MatrixXd & matr,
            Eigen::VectorXd & lastLine,
            Eigen::VectorXd & lactCol )   [pure virtual]
```

Генерация блока матрицы

Генерирует следующие компоненты матрицы:

- диагональный блок матрицы — влияние данного профиля на самого себя;
- нижнюю строку для матрицы для данного профиля;
- правый столбец матрицы для данного профиля.

Аргументы

out	matr	ссылка на генерируемую матрицу
out	lastLine	ссылка на нижнюю строку
out	lactCol	ссылка на правый столбец

Замещается в BoundaryConstLayerAver.

Граф вызова функции:

10.4.3.4 FillRhs()

```
virtual void Boundary::FillRhs (
            const Point2D & V0,
            Eigen::VectorXd & rhs,
            double * lastRhs,
            bool move,
            bool deform )   [pure virtual]
```

Заполнение в правой части влияния набегающего потока, следа и присоединенных слоев, действующих от самого себя

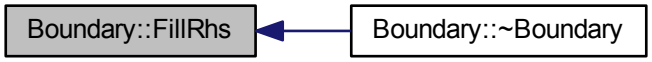
Заполняет блок правой части матрицы СЛАУ, обеспечивающей удовлетворение граничного условия, соответствующий данному профилю. Учитывается влияние набегающего потока, следа и присоединенных слоев, действующих от самого себя

Аргументы

in	V0	константная ссылка на вектор набегающего потока
out	rhs	ссылка на блок вектора правой части
out	lastRhs	указатель на последний элемент вектора правой части
in	move	является ли профиль подвижным
in	deform	является ли профиль деформируемым

Замещается в BoundaryConstLayerAver.

Граф вызова функции:



10.4.3.5 FillRhsFromOther()

```
virtual void Boundary::FillRhsFromOther (
            const Airfoil & otherAirfoil,
            Eigen::VectorXd & rhs )   [pure virtual]
```

Заполнение в правой части влияния присоединенных слоев, действующих на один профиль от другого

Заполняет блок правой части матрицы СЛАУ, обеспечивающий удовлетворение граничного условия, . Учитывается влияние присоединенных слоев от другого профиля

Аргументы

out	otherAirfoil	константная ссылка профиль, вклад от которого в правую часть требуется вычислить
out	rhs	ссылка на блок вектора правой части, соответствующего профилю, на который учитывается влияние

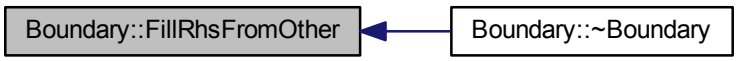
Необходимо сделать Пока считается, что граничные условия одинаковые

Предупреждения

Меняет rhs путем прибавления

Замещается в BoundaryConstLayerAver.

Граф вызова функции:



10.4.3.6 GetConvVelocityAtVirtualVortexes()

virtual void Boundary::GetConvVelocityAtVirtualVortexes (
std::vector< Point2D > & velo) const [pure virtual]

Вычисление конвективной скорости в точках расположения виртуальных вихрей

Вычисление производится в точках расположения виртуальных вихрей. Скорости находятся как скорости соответствующих точек профиля плюс половина интенсивности присоединенного вихревого слоя, умноженная на касательную к профилю на данной панели.

Аргументы

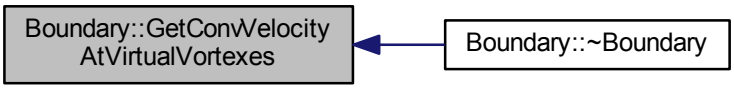
out	velo	ссылка на заполняемый список скоростей

Предупреждения

Массив velo заполняется путем присвоения, а не прибавления значений

Замещается в BoundaryConstLayerAver.

Граф вызова функции:



10.4.3.7 GetUnknownsSize()

virtual size_t Boundary::GetUnknownsSize ( ) const [pure virtual]

Возврат размерности вектора решения

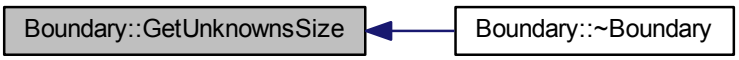
(без учета регуляризирующей переменной)

Возвращает

размерность вектора решения

Замещается в BoundaryConstLayerAver.

Граф вызова функции:



10.4.3.8 SolutionToFreeVortexSheetAndVirtualVortex()

virtual void Boundary::SolutionToFreeVortexSheetAndVirtualVortex (
const Eigen::VectorXd & sol) [pure virtual]

Пересчет решения на интенсивность вихревого слоя и на рождаемые вихри на конкретном профиле

1) Приводит решение к интенсивности вихревого слоя и записывает его в sheets.freeVortexSheet:

- если неизвестное — интенсивность вихря, то он "размазывается" по панели;
- если неизвестное — интенсивность слоя, то она передается непосредственно.

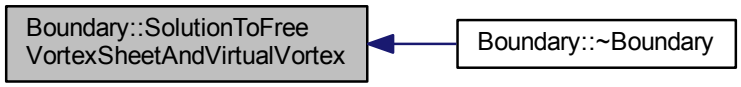
2) Приводит интенсивность вихревого слоя к рождаемым вихрям, а также вычисляет их положения

Аргументы

in	sol	вектор решения СЛАУ

Замещается в BoundaryConstLayerAver.

Граф вызова функции:



10.4.4 Данные класса

10.4.4.1 afl

const Airfoil& Boundary::afl

Константная ссылка на профиль
инициализируется автоматом в конструкторе

См. определение в файле Boundary.h строка 75

10.4.4.2 CC

const std::vector<Point2D>& Boundary::CC

Константная ссылка на вектор из начал (и концов) панелей

См. определение в файле Boundary.h строка 78

10.4.4.3 numberInPassport

const size_t Boundary::numberInPassport [protected]

Номер профиля в паспорте

См. определение в файле Boundary.h строка 69

10.4.4.4 oldSheets

Sheet Boundary::oldSheets

Слои на профиле с предыдущего шага

См. определение в файле Boundary.h строка 104

10.4.4.5 sheetDim

size_t Boundary::sheetDim

Размерность параметров каждого из слоев на каждой из панелей

Указывает, сколькими числами задается интенсивность каждого из слоев на каждой панели:

- 1 — одно число — задается только среднее значение;
- 2 — два числа — задается среднее значение и "наклон".

См. определение в файле Boundary.h строка 98

10.4.4.6 sheets

Sheet Boundary::sheets

Слои на профиле

См. определение в файле Boundary.h строка 101

10.4.4.7 virtualWake

VirtualWake Boundary::virtualWake

Виртуальный вихревой след конкретного профиля

См. определение в файле Boundary.h строка 84

10.4.4.8 vortexPerPanel

int Boundary::vortexPerPanel

Число вихрей, рождаемых на каждой панели профиля и формирующих виртуальный вихревой след

См. определение в файле Boundary.h строка 81

10.4.4.9 W

const World2D& Boundary::W [protected]

Константная ссылка на решаемую задачу

См. определение в файле Boundary.h строка 66

Объявления и описания членов классов находятся в файлах:

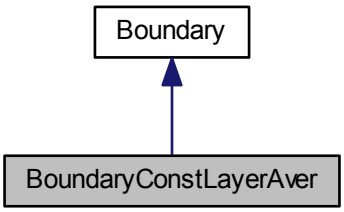
- Boundary/Boundary.h
- Boundary/Boundary.cpp

10.5 Класс BoundaryConstLayerAver

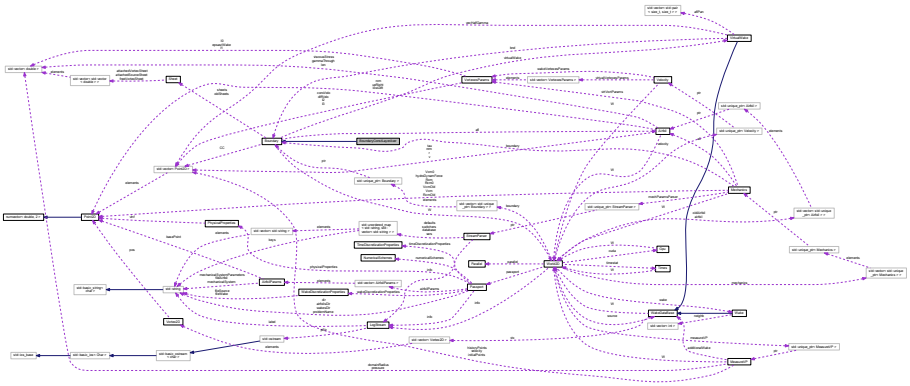
Класс, определяющий способ удовлетворения граничного условия на обтекаемом профиле

#include <BoundaryConstLayerAver.h>

Граф наследования:BoundaryConstLayerAver:



Граф связей класса BoundaryConstLayerAver:



Открытые члены

- BoundaryConstLayerAver (const World2D &W_, size_t numberInPassport_)

  Конструктор
- virtual ~BoundaryConstLayerAver ()

  Деструктор
- virtual void FillMatrixSelf (Eigen::MatrixXd &matr, Eigen::VectorXd &lastLine, Eigen::VectorXd &lactCol)

  Генерация блока матрицы
- virtual void FillMatrixFromOther (const Boundary &otherBoundary, Eigen::MatrixXd &matr)

  Генерация блока матрицы влияния от другого профиля того же типа
- virtual void FillRhs (const Point2D &V0, Eigen::VectorXd &rhs, double *lastRhs, bool move, bool deform)

  Заполнение в правой части влияния набегающего потока, следа и присоединенных слоев, действующих от самого себя
- virtual size_t GetUnknownsSize () const

  Возврат размерности вектора решения
- virtual void SolutionToFreeVortexSheetAndVirtualVortex (const Eigen::VectorXd &sol)

  Пересчет решения на интенсивность вихревого слоя и на рождаемые вихри на конкретном профиле
- virtual void GetWakeInfluence (std::vector< double > &wakeVelo) const

  Генерация вектора влияния вихревого следа на профиль
- virtual void GetConvVelocityToSetOfPoints (const std::vector< Vortex2D > &points, std::vector< Point2D > &velo) const

  Вычисление конвективных скоростей в наборе точек, вызываемых наличием завихренности и источников на профиле как от слоев
- virtual void GetConvVelocityToSetOfPointsFromVirtualVortexes (const WakeDataBase &points↩Db, std::vector< Point2D > &velo) const

  Вычисление конвективных скоростей в наборе точек, вызываемых наличием завихренности и источников на профиле как виртуальных вихрей
- virtual void GetConvVelocityAtVirtualVortexes (std::vector< Point2D > &velo) const

  Вычисление конвективной скорости в точках расположения виртуальных вихрей
- virtual void ComputeAttachedSheetsIntensity ()

  Вычисление интенсивностей присоединенного вихревого слоя и присоединенного слоя источников
- virtual void FillRhsFromOther (const Airfoil &otherAirfoil, Eigen::VectorXd &rhs)

  Заполнение в правой части влияния присоединенных слоев, действующих на один профиль от другого
- void VirtualWakeSynchronize ()

  MPI-синхронизация виртуального вихревого следа

Открытые атрибуты

- const Airfoil & afl
- const std::vector< Point2D > & CC

  Константная ссылка на вектор из начал (и концов) панелей
- int vortexPerPanel

  Число вихрей, рождаемых на каждой панели профиля и формирующих виртуальный вихревой след
- VirtualWake virtualWake

  Виртуальный вихревой след конкретного профиля
- size_t sheetDim

  Размерность параметров каждого из слоев на каждой из панелей
- Sheet sheets

  Слои на профиле
- Sheet oldSheets

  Слои на профиле с предыдущего шага

Защищенные данные

- const World2D & W

  Константная ссылка на решаемую задачу
- const size_t numberInPassport

  Номер профиля в паспорте

Закрытые члены

- double Alpha (const Point2D &p, const Point2D &s) const

  Вспомогательная функция вычисления угла между векторами
- double Lambda (const Point2D &p, const Point2D &s) const

  Вспомогательная функция вычисления логарифма отношения норм векторов
- Point2D Omega (const Point2D &a, const Point2D &b, const Point2D &c) const

  Вспомогательная функция вычисления величины $(\vec{a} \cdot \vec{b}) \cdot \vec{c} + (\vec{a} \times \vec{b}) \times \vec{c}$.

10.5.1 Подробное описание

Класс, определяющий способ удовлетворения граничного условия на обтекаемом профиле

Способ удовлетворения граничного условия:

- генерация вихревого слоя постоянной интенсивности на панелях;
- условие ортогональности невязки граничного условия константе (выполнение граничного условия в среднем по панели).

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле BoundaryConstLayerAver.h строка 61

10.5.2 Конструктор(ы)

10.5.2.1 BoundaryConstLayerAver()

```
BoundaryConstLayerAver::BoundaryConstLayerAver (
            const World2D & W_,
            size_t numberInPassport_ )   [inline]
```

Конструктор

Аргументы

in	W_	константная ссылка на решаемую задачу
in	numberIn↩ Passport_	номер профиля в паспорте задачи

См. определение в файле BoundaryConstLayerAver.h строка 103

```
103                                                    :
104        Boundary(W_, numberInPassport_, 1)
105    { };
```

10.5.2.2 ~BoundaryConstLayerAver()

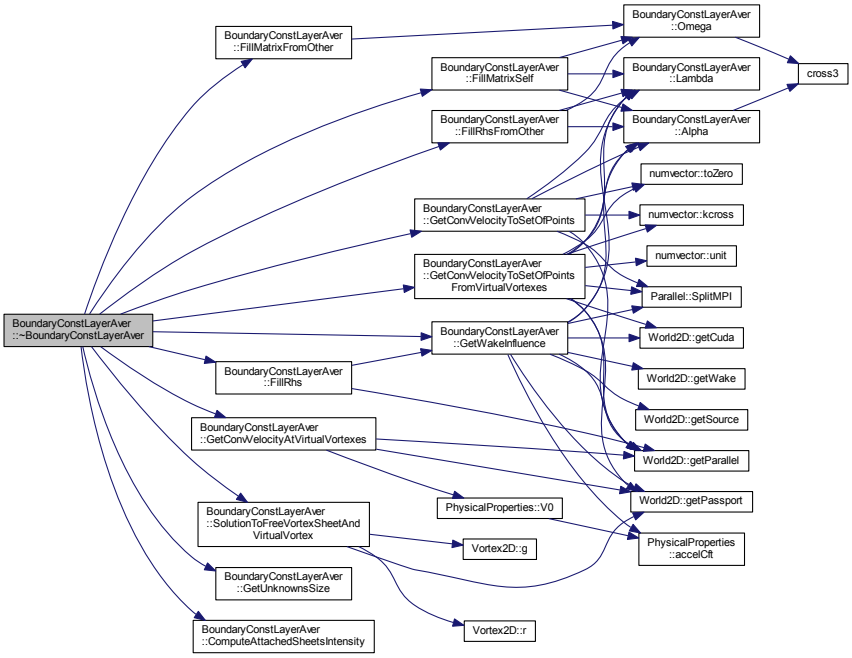
virtual BoundaryConstLayerAver::~BoundaryConstLayerAver ( ) [inline], [virtual]

Деструктор

См. определение в файле BoundaryConstLayerAver.h строка 108

```
108 {};
```

Граф вызовов:



10.5.3 Методы

10.5.3.1 Alpha()

```
double BoundaryConstLayerAver::Alpha (
            const Point2D & p,
            const Point2D & s ) const   [inline], [private]
```

Вспомогательная функция вычисления угла между векторами

Аргументы

in	p	константная ссылка на первый вектор
in	s	константная ссылка на второй вектор

Возвращает

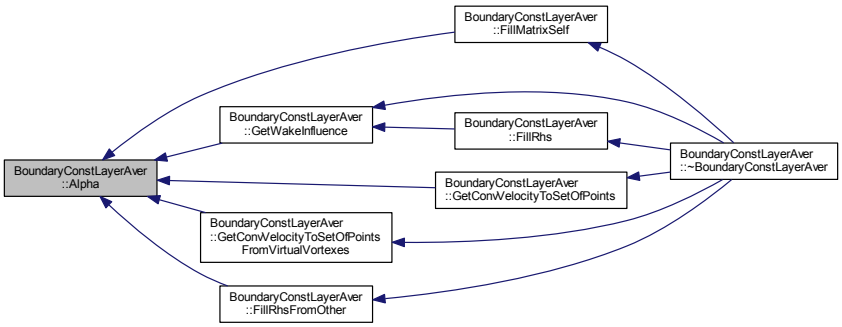
угол между векторами в диапазоне $(-\pi; \pi]$

См. определение в файле BoundaryConstLayerAver.h строка 69

```
70    {
71        return atan2(cross3(p, s), p*s);
72    }
```

Граф вызовов:

Граф вызова функции:



10.5.3.2 ComputeAttachedSheetsIntensity()

void BoundaryConstLayerAver::ComputeAttachedSheetsIntensity ( ) [virtual]

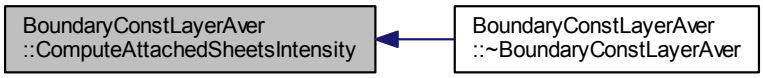
Вычисление интенсивностей присоединенного вихревого слоя и присоединенного слоя источников

Замещает Boundary.

См. определение в файле BoundaryConstLayerAver.cpp строка 731

```
{
    for (size_t i = 0; i < sheets.attachedVortexSheet.size(); ++i)
    {
        oldSheets.attachedVortexSheet[i][0] = sheets.attachedVortexSheet[i][0];
        oldSheets.attachedSourceSheet[i][0] = sheets.attachedVortexSheet[i][0];
    }

    for (size_t i = 0; i < sheets.attachedVortexSheet.size(); ++i)
    {
        sheets.attachedVortexSheet[i][0] = afl.v[i] * afl.tau[i];
        sheets.attachedSourceSheet[i][0] = afl.v[i] * afl.nrm[i];
    }
}//ComputeAttachedSheetsIntensity()
```

Граф вызова функции:



10.5.3.3 FillMatrixFromOther()

void BoundaryConstLayerAver::FillMatrixFromOther (
const Boundary & otherBoundary,
Eigen::MatrixXd & matr) [virtual]

Генерация блока матрицы влияния от другого профиля того же типа

Генерирует блок матрицы влияния от другого профиля того же типа

Аргументы

in	otherBoundary	константная ссылка на граничное условие на втором профиле
out	matr	ссылка на генерируемый блок матрицы

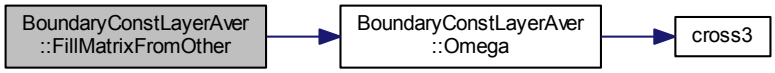
Необходимо сделать Пока считается, что граничные условия одинаковые

Замещает Boundary.

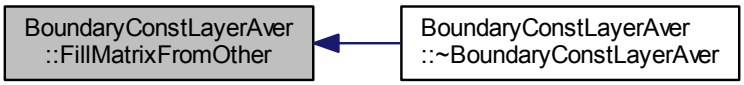
См. определение в файле BoundaryConstLayerAver.cpp строка 166

```
{
    size_t np = afl.np;
    size_t npOther = otherBoundary.afl.np;

    //Panel vectors
    std::vector<Point2D> dd;
    for (size_t i = 0; i < np; ++i)
        dd.push_back(afl.tau[i] * afl.len[i]);

    std::vector<Point2D> ddOther;
    for (size_t j = 0; j < npOther; ++j)
        ddOther.push_back(otherBoundary.afl.tau[j] * otherBoundary.afl.len[j]);

    //auxillary scalars
    numvector<double, 3> alpha, lambda;

    //auxillary vectors
    Point2D p1, s1, p2, s2, i00;
    numvector<Point2D, 3> v;

    for (size_t i = 0; i < np; ++i)
    for (size_t j = 0; j < npOther; ++j)
    {
        const Point2D& di = dd[i];
        const Point2D& dj = ddOther[j];

        const Point2D& taui = afl.tau[i];
        const Point2D& tauj = otherBoundary.afl.tau[j];

        p1 = CC[i + 1] - otherBoundary.CC[j + 1];
        s1 = CC[i + 1] - otherBoundary.CC[j];
        p2 = CC[i] - otherBoundary.CC[j + 1];
        s2 = CC[i] - otherBoundary.CC[j];

        alpha = { \
            Alpha(s2, s1), \
            Alpha(s2, p1), \
            Alpha(p1, p2) \
        };

        lambda = { \
            Lambda(s2, s1), \
            Lambda(s2, p1), \
            Lambda(p1, p2) \
        };

        v = { Omega(s1, taui, tauj), -Omega(di, taui, tauj), Omega(p2, taui, tauj) };

        i00 = IDPI / afl.len[i] * ((alpha[0] * v[0] + alpha[1] * v[1] + alpha[2] * v[2]) + (lambda[0] * v[0] + lambda[1] * v[1] + lambda[2] * v[2]).kcross());

        //i00 = IDPI / afl.len[i] * (-(alpha[0] * v[0] + alpha[1] * v[1] + alpha[2] * v[2]).kcross() + (lambda[0] * v[0] + lambda[1] * v[1] + lambda[2] * v[2]));


        matr(i, j) = i00 * afl.tau[i];
    }

}//FillMatrixFromOther(...)
```

Граф вызовов:



Граф вызова функции:



10.5.3.4 FillMatrixSelf()

void BoundaryConstLayerAver::FillMatrixSelf (
Eigen::MatrixXd & matr,
Eigen::VectorXd & lastLine,
Eigen::VectorXd & lactCol) [virtual]

Генерация блока матрицы

Генерирует следующие компоненты матрицы:

- диагональный блок матрицы — влияние данного профиля на самого себя;
- нижнюю строку для матрицы для данного профиля;
- правый столбец матрицы для данного профиля.

Аргументы

out	matr	ссылка на генерируемую матрицу
out	lastLine	ссылка на нижнюю строку
out	lactCol	ссылка на правый столбец

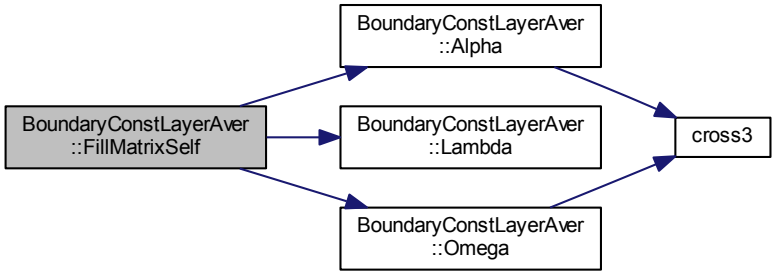
Замещает Boundary.

См. определение в файле BoundaryConstLayerAver.cpp строка 94

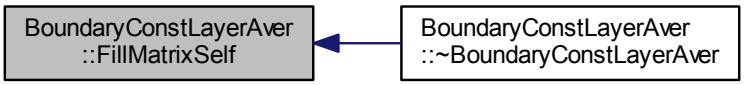
```
95 {
96     size_t np = afl.np;
97
98     //Panel vectors
99     std::vector<Point2D> dd;
100     for (size_t i = 0; i < np; ++i)
101         dd.push_back(afl.tau[i] * afl.len[i]);
102
103     for (size_t i = 0; i < np; ++i)
104     {
105         lactCol(i) = 1.0;
106         lastLine(i) = afl.len[i];
107     }
108
109     //auxillary scalars
110     numvector<double, 3> alpha, lambda;
```

```

    //auxillary vectors
    Point2D p1, s1, p2, s2, i00;
    numvector<Point2D, 3> v;

    for (size_t i = 0; i < np; ++i)
    for (size_t j = 0; j < np; ++j)
    {
        if (i != j)
        {
            const Point2D& di = dd[i];
            const Point2D& dj = dd[j];

            const Point2D& taui = afl.tau[i];
            const Point2D& tauj = afl.tau[j];

            p1 = CC[i + 1] - CC[j + 1];
            s1 = CC[i + 1] - CC[j];
            p2 = CC[i] - CC[j + 1];
            s2 = CC[i] - CC[j];

            alpha = { \
                afl.isAfter(j, i) ? 0.0 : Alpha(s2, s1), \
                Alpha(s2, p1), \
                afl.isAfter(i, j) ? 0.0 : Alpha(p1, p2) \
            };

            lambda = { \
                afl.isAfter(j, i) ? 0.0 : Lambda(s2, s1), \
                Lambda(s2, p1), \
                afl.isAfter(i, j) ? 0.0 : Lambda(p1, p2) \
            };

            v = { Omega(s1, taui, tauj), -Omega(di, taui, tauj), Omega(p2, taui, tauj) };

            i00 = IDPI / afl.len[i] * ((alpha[0] * v[0] + alpha[1] * v[1] + alpha[2] * v[2]) + (
lambda[0] * v[0] + lambda[1] * v[1] + lambda[2] * v[2]).kcross());

            //i00 = IDPI / afl.len[i] * (-(alpha[0] * v[0] + alpha[1] * v[1] + alpha[2] * v[2]).kcross() +
 (lambda[0] * v[0] + lambda[1] * v[1] + lambda[2] * v[2]));


            matr(i, j) = i00 * afl.tau[i];
        }
    }

    for (size_t i = 0; i < np; ++i)
    {
        // (afl.tau[i] ^ afl.nrm[i]) для учета внешней нормали
        matr(i, i) = 0.5 * (afl.tau[i] ^ afl.nrm[i]);
    }
}//FillMatrixSelf(...)
```

Граф вызовов:



Граф вызова функции:



10.5.3.5 FillRhs()

```
void BoundaryConstLayerAver::FillRhs (
            const Point2D & V0,
            Eigen::VectorXd & rhs,
            double * lastRhs,
            bool move,
            bool deform )   [virtual]
```

Заполнение в правой части влияния набегающего потока, следа и присоединенных слоев, действующих от самого себя

Заполняет блок правой части матрицы СЛАУ, обеспечивающей удовлетворение граничного условия, соответствующий данному профилю. Учитывается влияние набегающего потока, следа и присоединенных слоев, действующих от самого себя

Аргументы

in	V0	константная ссылка на вектор набегающего потока
out	rhs	ссылка на блок вектора правой части
out	lastRhs	указатель на последний элемент вектора правой части
in	move	является ли профиль подвижным
in	deform	является ли профиль деформируемым

Необходимо сделать Это должно быть закомментировано или нет?

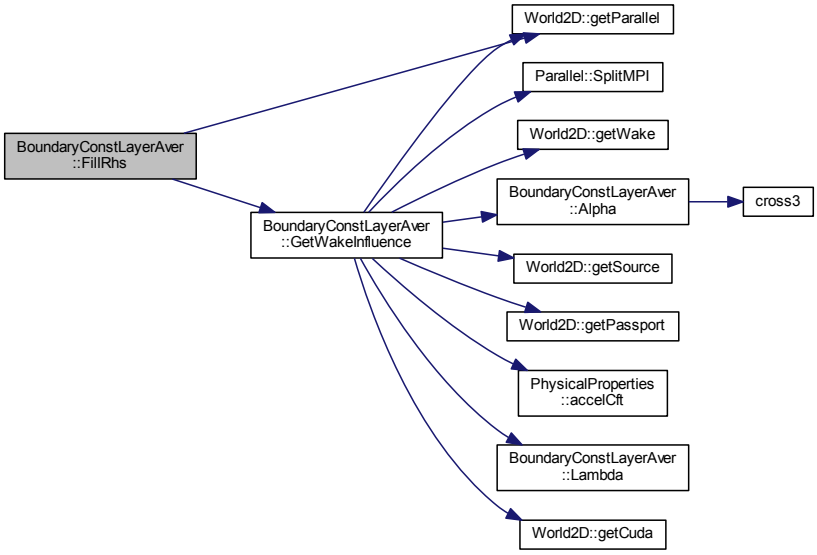
Замещает Boundary.

См. определение в файле BoundaryConstLayerAver.cpp строка 621

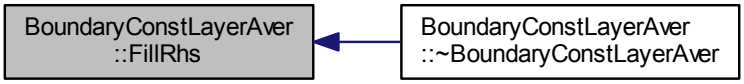
```
622 {
623     std::vector<double> wakeVelo;
624
625 #if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_RHS))
626     GPUGetWakeInfluence(wakeVelo);
627 #else
628     GetWakeInfluence(wakeVelo);
629 #endif
630
```

```
    if (W.getParallel().myidWork == 0)
    for (size_t i = 0; i < afl.np; ++i)
    {
        rhs(i) = -afl.tau[i] * V0 - wakeVelo[i];

        /*
        //влияние присоединенных слоев от самого себя
        if (move || deform)
        for (size_t j = 0; j < afl.np; j++)
        {
            if (i != j)
            {
                Point2D di = afl.r[i + 1] - afl.r[i];
                Point2D dj = afl.r[j + 1] - afl.r[j];
                Point2D s1 = afl.r[i + 1] - afl.r[j];
                Point2D s2 = afl.r[i] - afl.r[j];
                Point2D p1 = afl.r[i + 1] - afl.r[j + 1];
                Point2D p2 = afl.r[i] - afl.r[j + 1];

                double a1 = Alpha(s2, s1);
                double a2 = Alpha(s2, p1);
                double a3 = Alpha(p1, p2);

                double lambda1 = Lambda(s1, s2);
                double lambda2 = Lambda(p1, s2);
                double lambda3 = Lambda(p2, p1);

                Point2D v1 = Omega(s1, afl.tau[i], afl.tau[j]);
                Point2D v2 = -Omega(di, afl.tau[i], afl.tau[j]);
                Point2D v3 = Omega(p2, afl.tau[i], afl.tau[j]);

                if ((i == j + 1) || ((i == 0) && (j == afl.np-1)))
                {
                    a3 = 0.0; lambda3 = 0.0;
                }
                else if ((j == i + 1) || ((j == 0) && (i == afl.np - 1)))
                {
                    a1 = 0.0; lambda1 = 0.0;
                }

                rhs(i) += -IDPI / afl.len[i] * sheets.attachedVortexSheet[j][0] * afl.tau[i] * (((-(a1 * v1 + a2 * v2 + a3 * v3).kcross()) + lambda1 * v1 + lambda2 * v2 + lambda3 * v3).kcross());
                rhs(i) += -IDPI / afl.len[i] * sheets.attachedSourceSheet[j][0] * afl.tau[i] * (((-(a1 * v1 + a2 * v2 + a3 * v3).kcross()) + lambda1 * v1 + lambda2 * v2 + lambda3 * v3));

            }//if (i != j)
        }//for j
        // * (afl.tau[i] ^ afl.nrm[i]) для учета внешней нормали
        rhs(i) += 0.5 *  sheets.attachedVortexSheet[i][0] * (afl.tau[i] ^ afl.nrm[i]);

        */
    }//for i

    if (W.getParallel().myidWork == 0)
    {
        *lastRhs = 0.0;

        for (size_t q = 0; q < afl.gammaThrough.size(); ++q)
            //for each (double g in afl.gammaThrough)
        {
            *lastRhs += afl.gammaThrough[q];
            //*lastRhs += g;
        }
    }
}//FillRhs(...)
```

Граф вызовов:



Граф вызова функции:



10.5.3.6 FillRhsFromOther()

void BoundaryConstLayerAver::FillRhsFromOther (
const Airfoil & otherAirfoil,
Eigen::VectorXd & rhs) [virtual]

Заполнение в правой части влияния присоединенных слоев, действующих на один профиль от другого

Заполняет блок правой части матрицы СЛАУ, обеспечивающий удовлетворение граничного условия, . Учитывается влияние присоединенных слоев от другого профиля

Аргументы

out	otherAirfoil	константная ссылка профиль, вклад от которого в правую часть требуется вычислить
out	rhs	ссылка на блок вектора правой части, соответствующего профилю, на который учитывается влияние

Необходимо сделать Пока считается, что граничные условия одинаковые

Предупреждения

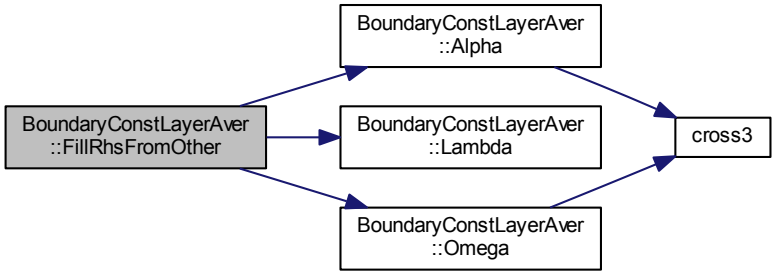
Меняет rhs путем прибавления

Замещает Boundary.

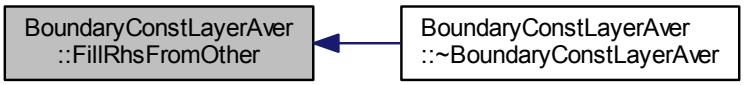
См. определение в файле BoundaryConstLayerAver.cpp строка 698

```
699 {
700     for (size_t i = 0; i < afl.np; ++i)
701     {
702         for (size_t j = 0; j < otherAirfoil.np; j++)
703         {
704             Point2D di = afl.r[i + 1] - afl.r[i];
705             Point2D dj = otherAirfoil.r[j + 1] - otherAirfoil.r[j];
706             Point2D s1 = afl.r[i + 1] - otherAirfoil.r[j];
707             Point2D s2 = afl.r[i] - otherAirfoil.r[j];
708             Point2D p1 = afl.r[i + 1] - otherAirfoil.r[j + 1];
709             Point2D p2 = afl.r[i] - otherAirfoil.r[j + 1];
710
711             double a1 = Alpha(s2, s1);
712             double a2 = Alpha(s2, p1);
713             double a3 = Alpha(p1, p2);
714
715             double lambda1 = Lambda(s1, s2);
716             double lambda2 = Lambda(p1, s2);
717             double lambda3 = Lambda(p2, p1);
718
719             Point2D v1 = Omega(s1, afl.tau[i], otherAirfoil.
    tau[j]);
720             Point2D v2 = -Omega(di, afl.tau[i], otherAirfoil.
    tau[j]);
721             Point2D v3 = Omega(p2, afl.tau[i], otherAirfoil.
    tau[j]);
722
723             rhs(i) += -IDPI / afl.len[i] * sheets.
    attachedVortexSheet[j][0] * afl.tau[i] * (((-(a1 * v1 + a2 * v2 + a3 * v3).kcross(
    )) + lambda1 * v1 + lambda2 * v2 + lambda3 * v3).kcross());
724             rhs(i) += -IDPI / afl.len[i] * sheets.
    attachedSourceSheet[j][0] * afl.tau[i] * (((-(a1 * v1 + a2 * v2 + a3 * v3).kcross(
    )) + lambda1 * v1 + lambda2 * v2 + lambda3 * v3));
725         }//for j
726     }//for i
727 }//FillRhsFromOther(...)
```

Граф вызовов:



Граф вызова функции:



10.5.3.7 GetConvVelocityAtVirtualVortexes()

void BoundaryConstLayerAver::GetConvVelocityAtVirtualVortexes (
std::vector< Point2D > & velo) const [virtual]

Вычисление конвективной скорости в точках расположения виртуальных вихрей

Вычисление производится в точках расположения виртуальных вихрей. Скорости находятся как скорости соответствующих точек профиля плюс половина интенсивности присоединенного вихревого слоя, умноженная на касательную к профилю на данной панели.

Аргументы

out	velo	ссылка на заполняемый список скоростей

Предупреждения

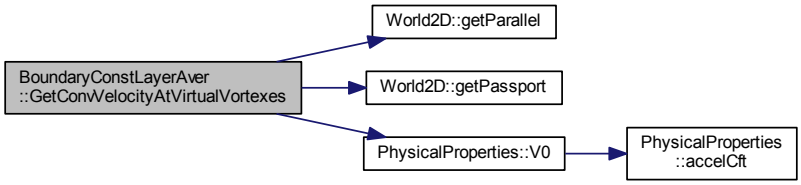
Массив velo заполняется путем присвоения, а не прибавления значений

Замещает Boundary.

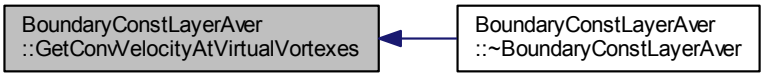
См. определение в файле BoundaryConstLayerAver.cpp строка 601

```
602 {
603     const int& id = W.getParallel().myidWork;
604     std::vector<Point2D>& Vel = velo;
605
606     //Скорости виртуальных вихрей
607     if (id == 0)
608     {
609 #pragma omp parallel for default(none) shared(Vel)
610         for (int i = 0; i < Vel.size(); ++i)
611             Vel[i] = afl.v[virtualWake.aflPan[i].second] \
612                 + virtualWake.vecHalfGamma[i] \
613                 - W.getPassport().physicalProperties.
    V0();
614     }
615 }
```

Граф вызовов:



Граф вызова функции:



10.5.3.8 GetConvVelocityToSetOfPoints()

void BoundaryConstLayerAver::GetConvVelocityToSetOfPoints (
const std::vector< Vortex2D > & points,
std::vector< Point2D > & velo) const [virtual]

Вычисление конвективных скоростей в наборе точек, вызываемых наличием завихренности и источников на профиле как от слоев

Вычисляет конвективные скорости в наборе точек, которые вызваны влиянием завихренности и источников на профиле как от слоев

Аргументы

in	points	константная ссылка на набор точек, в которых вычисляются скорости
out	velo	ссылка на вектор скоростей, которые приобретают точки из-за влияния завихренности и источников на профиле

Предупреждения

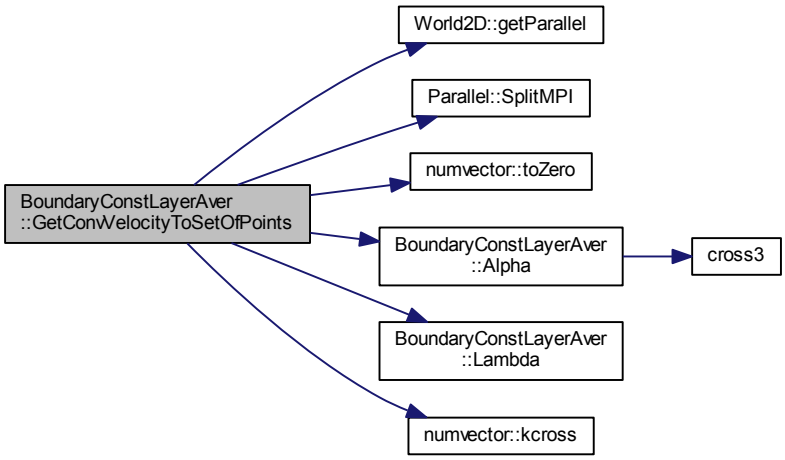
velo — накапливается!
Использует OMP, MPI

Замещает Boundary.

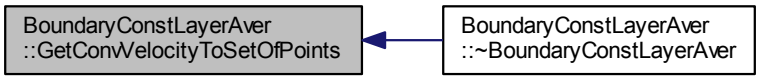
См. определение в файле BoundaryConstLayerAver.cpp строка 395

```
{
    std::vector<Point2D> selfVelo;

    size_t np = afl.np;

    int id = W.getParallel().myidWork;

    parProp par = W.getParallel().SplitMPI(points.size());

    std::vector<Vortex2D> locPoints;
    locPoints.resize(par.myLen);

    MPI_Scatterv(const_cast<std::vector<Vortex2D>&>(points).data(), par.len.data(), par.disp.data(), Vortex2D::mpiVortex2D, \
        locPoints.data(), par.myLen, Vortex2D::mpiVortex2D, 0, W.getParallel().commWork);

    std::vector<Point2D> locVelo;
    locVelo.resize(par.myLen);

    //Локальные переменные для цикла
    Point2D velI;
    Point2D tempVel;

#pragma omp parallel for default(none) shared(locVelo, locPoints, par) private(velI, tempVel)
    for (int i = 0; i < par.myLen; ++i)
    {
        velI.toZero();

        const Point2D& posI = locPoints[i].r();

        for (size_t j = 0; j < afl.np; ++j)
        {
            const Point2D& posJ0 = afl.r[j];
            const Point2D& posJ1 = afl.r[j + 1];
            const Point2D& tau = afl.tau[j];

            double gamJ = sheets.freeVortexSheet[j][0];

            Point2D s = posI - posJ0;
            Point2D p = posI - posJ1;

            double alpha = Alpha(p, s);

            double lambda = Lambda(p, s);

            tempVel = alpha* tau + lambda * tau.kcross();
            tempVel *= gamJ;
            velI += tempVel;
        } //for j

        velI *= IDPI;
        locVelo[i] = velI;
    }

    if (id == 0)
        selfVelo.resize(points.size());

    MPI_Gatherv(locVelo.data(), par.myLen, Point2D::mpiPoint2D, selfVelo.data(), par.len.data(), par.disp.data(), Point2D::mpiPoint2D, 0, W.getParallel().commWork);

    if (id == 0)
    for (size_t i = 0; i < velo.size(); ++i)
        velo[i] += selfVelo[i];
}//GetVelocityToSetOfPoints(...)
```

Граф вызовов:



Граф вызова функции:



10.5.3.9 GetConvVelocityToSetOfPointsFromVirtualVortexes()

void BoundaryConstLayerAver::GetConvVelocityToSetOfPointsFromVirtualVortexes (
const WakeDataBase & pointsDb,
std::vector< Point2D > & velo) const [virtual]

Вычисление конвективных скоростей в наборе точек, вызываемых наличием завихренности и источников на профиле как виртуальных вихрей

Вычисляет конвективные скорости в наборе точек, которые вызваны влиянием завихренности и источников на профиле как виртуальных вихрей

Аргументы

in	pointsDb	константная ссылка на базу данных вихрей, в точках которых вычисляются скорости
out	velo	ссылка на вектор скоростей, которые приобретают точки из-за влияния завихренности и источников на профиле

Предупреждения

velo — накапливается!
Использует OMP, MPI

Необходимо сделать Тут надо разобраться, как должно быть...

Необходимо сделать сделать if(move || deform)

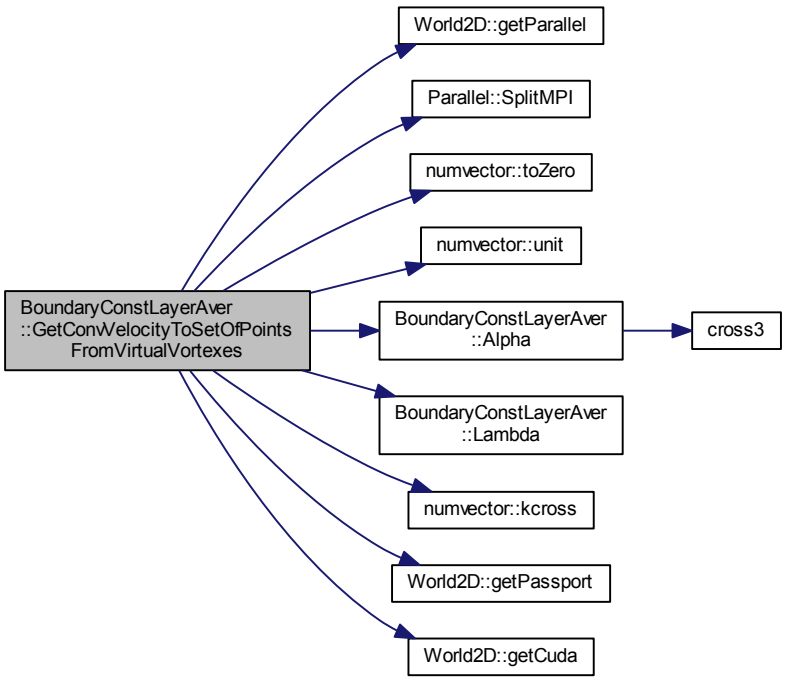
Замещает Boundary.

См. определение в файле BoundaryConstLayerAver.cpp строка 461

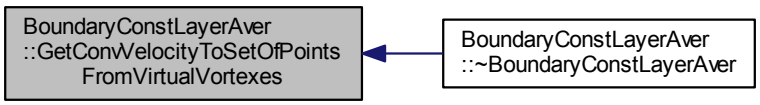
```
462 {
463     std::vector<Point2D> selfVelo;
464
465     size_t np = afl.np;
466
467     int id = W.getParallel().myidWork;
468
469     parProp par = W.getParallel().SplitMPI(pointsDb.
    vtx.size());
470
471     //синхронизация свободного вихревого слоя
472     std::vector<double> freeVortexSheetGamma;
473     int sz;
474
475     if (id == 0)
476     {
477         sz = (int)sheets.freeVortexSheet.size();
478         freeVortexSheetGamma.resize(sz, 0.0);
479         for (size_t j = 0; j < sheets.freeVortexSheet.size(); ++j)
480             freeVortexSheetGamma[j] = sheets.freeVortexSheet[j][0];
481     }
482     MPI_Bcast(&sz, 1, MPI_INT, 0, W.getParallel().commWork);
483     if (id != 0)
484         freeVortexSheetGamma.resize(sz, 0.0);
485
486     MPI_Bcast(freeVortexSheetGamma.data(), (int)sz, MPI_DOUBLE, 0, W.
    getParallel().commWork);
487     //свободный вихревой слой синхронизирован
488
489
490
491     std::vector<Vortex2D> locPoints;
492     locPoints.resize(par.myLen);
493
494     MPI_Scatterv(const_cast<std::vector<Vortex2D>&>(pointsDb.vtx).data(), par.
    len.data(), par.disp.data(), Vortex2D::mpiVortex2D, \
495         locPoints.data(), par.myLen, Vortex2D::mpiVortex2D, 0,
    W.getParallel().commWork);
496
497     double cft = IDPI;
498
499     std::vector<Point2D> locConvVelo;
500     locConvVelo.resize(par.myLen);
501
502 #pragma warning (push)
503 #pragma warning (disable: 4101)
504     //Локальные переменные для цикла
505     Point2D velI;
506     Point2D tempVel;
507     //double dst2eps, dst2;
508 #pragma warning (pop)
509
510 #pragma omp parallel for default(none) shared(locConvVelo, locPoints, cft, par, freeVortexSheetGamma,
     std::cout) private(velI, tempVel/*, dst2, dst2eps*/)
511     for (int i = 0; i < par.myLen; ++i)
512     {
513         velI.toZero();
514
515         const Point2D& posI = locPoints[i].r();
516
519         for (size_t j = 0; j < sheets.attachedVortexSheet.size(); j++)
520         {
521             Point2D dj = afl.r[j + 1] - afl.r[j];
522             Point2D tauj = dj.unit();
523
```

```
            Point2D s = posI - afl.r[j];
            Point2D p = posI - afl.r[j + 1];

            double a = Alpha(p, s);

            double lambda = Lambda(p, s);

            velI += (freeVortexSheetGamma[j] * (-a * tauj.kcross() + lambda * tauj)).kcross();
            velI += (sheets.attachedVortexSheet[j][0] * (-a * tauj.kcross() + lambda * tauj)).kcross();
            velI += (sheets.attachedSourceSheet[j][0] * (-a * tauj.kcross() + lambda * tauj));
        }//for j

        velI *= cft;
        locConvVelo[i] = velI;
    }//for i

    if (id == 0)
        selfVelo.resize(pointsDb.vtx.size());

    MPI_Gatherv(locConvVelo.data(), par.myLen, Point2D::mpiPoint2D, selfVelo.data(), par.len.data(), par.disp.data(), Point2D::mpiPoint2D, 0, W.getParallel().commWork);

    if (id == 0)
    for (size_t i = 0; i < velo.size(); ++i)
        velo[i] += selfVelo[i];
}//GetVelocityToSetOfPointsFromVirtualVortexes(...)
```

Граф вызовов:



Граф вызова функции:



10.5.3.10 GetUnknownsSize()

size_t BoundaryConstLayerAver::GetUnknownsSize ( ) const [virtual]

Возврат размерности вектора решения

(без учета регуляризирующей переменной)

Возвращает

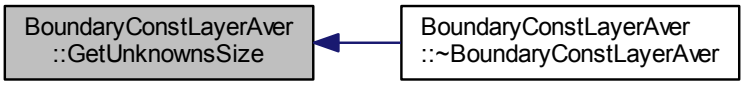
размерность вектора решения

Замещает Boundary.

См. определение в файле BoundaryConstLayerAver.cpp строка 44

```
45 {
46     return afl.np;
47 }//GetUnknownsSize()
```

Граф вызова функции:



10.5.3.11 GetWakeInfluence()

void BoundaryConstLayerAver::GetWakeInfluence (
std::vector< double > & wakeVelo) const [virtual]

Генерация вектора влияния вихревого следа на профиль

Генерирует вектор влияния вихревого следа на профиль, используемый затем для расчета вектора правой части.

Аргументы

out	wakeVelo	ссылка на вектор влияния вихревого следа на профиль

Предупреждения

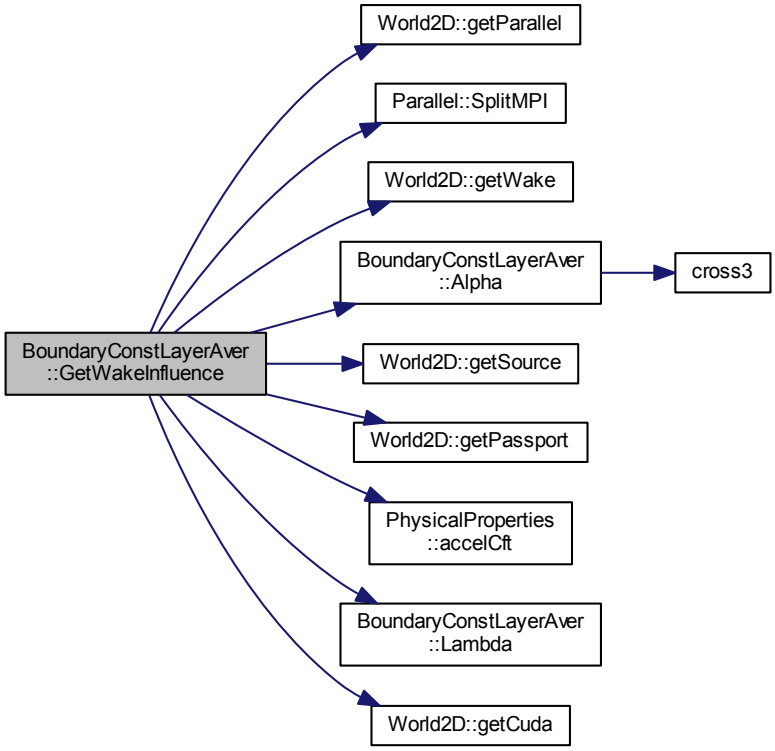
Использует OMP, MPI

Замещает Boundary.

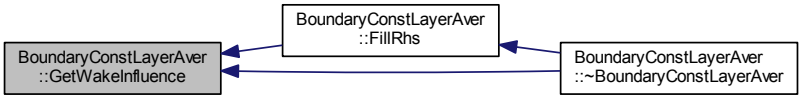
См. определение в файле BoundaryConstLayerAver.cpp строка 228

```
229 {
230     size_t np = afl.np;
231     int id = W.getParallel().myidWork;
232     parProp par = W.getParallel().SplitMPI(np);
233
234     std::vector<double> locVeloWake;
235     locVeloWake.resize(par.myLen);
236
237     //локальные переменные для цикла
238     double velI = 0.0;
239     double tempVel = 0.0;
240
241 #pragma omp parallel for default(none) shared(locVeloWake, par) private(velI, tempVel)
242     for (int i = 0; i < par.myLen; ++i)
243     {
244         velI = 0.0;
245
246         const Point2D& posI0 = CC[par.myDisp + i];
247         const Point2D& posI1 = CC[par.myDisp + i + 1];
248         const Point2D& tau = afl.tau[par.myDisp + i];
249
250         for (size_t j = 0; j < W.getWake().vtx.size(); ++j)
251         {
252             const Point2D& posJ = W.getWake().vtx[j].r();
253             const double& gamJ = W.getWake().vtx[j].g();
254
255             Point2D s = posJ - posI0;
256             Point2D p = posJ - posI1;
257
258             double alpha = Alpha(p, s);
259             //double lambda = Lambda(p, s); //не нужна для касательной
260
261             tempVel = gamJ * alpha;
262             velI -= tempVel;
263         }
264
265         for (size_t j = 0; j < W.getSource().vtx.size(); ++j)
266         {
267             const Point2D& posJ = W.getSource().vtx[j].r();
268             const double& gamJ = W.getPassport().physicalProperties.
    accelCft() * W.getSource().vtx[j].g();
269
270             Point2D s = posJ - posI0;
271             Point2D p = posJ - posI1;
272
273             //double alpha = Alpha(p, s); //не нужна для касательной
274             double lambda = Lambda(p, s);
275
276             tempVel = gamJ * lambda;
277             velI -= tempVel;
278         }
279
280         velI *= IDPI / afl.len[par.myDisp + i];
281         locVeloWake[i] = velI;
282     }
283
284     if (id == 0)
285         wakeVelo.resize(np);
286
287     MPI_Gatherv(locVeloWake.data(), par.myLen, MPI_DOUBLE, wakeVelo.data(), par.
    len.data(), par.disp.data(), MPI_DOUBLE, 0, W.getParallel().
    commWork);
288 }//GetWakeInfluence(...)
```

Граф вызовов:



Граф вызова функции:



10.5.3.12 Lambda()

```
double BoundaryConstLayerAver::Lambda (
            const Point2D & p,
            const Point2D & s ) const    [inline], [private]
```

Вспомогательная функция вычисления логарифма отношения норм векторов

Аргументы

in	p	константная ссылка на первый вектор
in	s	константная ссылка на второй вектор

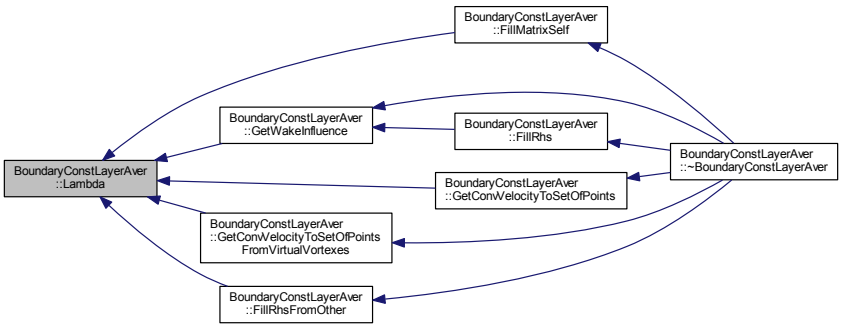
Возвращает

логарифм отношения норм векторов

См. определение в файле BoundaryConstLayerAver.h строка 79

```
80    {
81        return 0.5*log((s*s) / (p*p));
82    }
```

Граф вызова функции:



10.5.3.13 Omega()

```
Point2D BoundaryConstLayerAver::Omega (
            const Point2D & a,
            const Point2D & b,
            const Point2D & c ) const    [inline], [private]
```

Вспомогательная функция вычисления величины $(\vec{a} \cdot \vec{b}) \cdot \vec{c} + (\vec{a} \times \vec{b}) \times \vec{c}$.

Для оптимизации все векторы считаются двумерными

Аргументы

in	a	константная ссылка на первый вектор
in	b	константная ссылка на второй вектор
in	c	константная ссылка на третий вектор

Возвращает

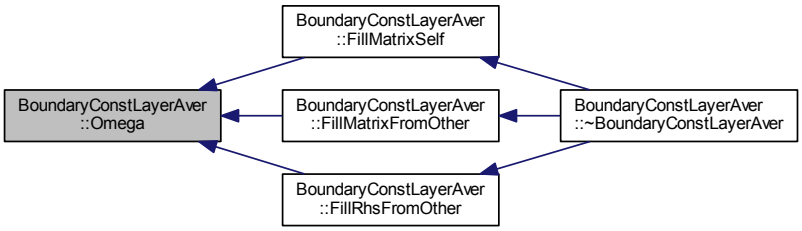
логарифм отношения норм векторов

См. определение в файле BoundaryConstLayerAver.h строка 92

```
93    {
94        return (a * b) * c + (Point2D({ -c[1], c[0] }))*cross3(a, b);
95    }
```

Граф вызовов:

Граф вызова функции:



10.5.3.14 SolutionToFreeVortexSheetAndVirtualVortex()

void BoundaryConstLayerAver::SolutionToFreeVortexSheetAndVirtualVortex (
const Eigen::VectorXd & sol) [virtual]

Пересчет решения на интенсивность вихревого слоя и на рождаемые вихри на конкретном профиле

1) Приводит решение к интенсивности вихревого слоя и записывает его в sheets.freeVortexSheet:

- если неизвестное — интенсивность вихря, то он "размазывается" по панели;
- если неизвестное — интенсивность слоя, то она передается непосредственно.

2) Приводит интенсивность вихревого слоя к рождаемым вихрям, а также вычисляет их положения

Аргументы

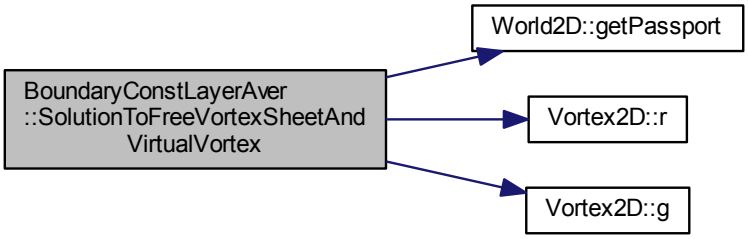
in	sol	вектор решения СЛАУ

Замещает Boundary.

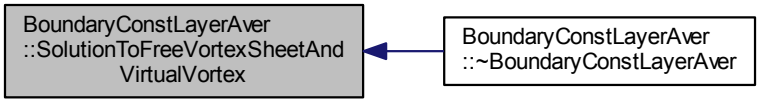
См. определение в файле BoundaryConstLayerAver.cpp строка 51

```
52 {
53     Vortex2D virtVort;
54     Point2D midNorm;
55
56     double delta = W.getPassport().wakeDiscretizationProperties.
   delta;
57
58     int nVortPerPan = W.getPassport().wakeDiscretizationProperties.
   vortexPerPanel;
59
60     //Очистка и резервирование памяти
61     virtualWake.vecHalfGamma.clear();
62     virtualWake.vecHalfGamma.reserve(afl.np * nVortPerPan);
63
64     //Очистка и резерваирование памяти
65     virtualWake.aflPan.clear();
66     virtualWake.aflPan.reserve(afl.np * nVortPerPan);
67
68     //Очистка и резерваирование памяти
69     virtualWake.vtx.reserve(afl.np * nVortPerPan);
70
71     for (size_t i = 0; i < afl.np ; ++i)
72     {
73         midNorm = afl.nrm[i] * delta;
74         Point2D dr = 1.0 / nVortPerPan * (afl.r[i + 1] - afl.r[i]);
75
76         for (int j = 0; j < nVortPerPan; j++)
77         {
78             virtVort.r() = afl.r[i] + dr * (j * 1.0 + 0.5) + midNorm;
79             virtVort.g() = sol(i) * afl.len[i] / nVortPerPan;
80             virtualWake.vtx.push_back(virtVort);
81
82             virtualWake.vecHalfGamma.push_back(0.5 * sol(i)  *
   afl.tau[i]);
83             virtualWake.aflPan.push_back({ numberInPassport, i });
84         }
85     }
86
87     for (size_t j = 0; j < afl.np; ++j)
88         sheets.freeVortexSheet[j][0] = sol(j);
89
90 }//SolutionToFreeVortexSheetAndVirtualVortex(...)
```

Граф вызовов:



Граф вызова функции:



10.5.4 Данные класса

10.5.4.1 afl

const Airfoil& Boundary::afl [inherited]

Константная ссылка на профиль
инициализируется автоматом в конструкторе

См. определение в файле Boundary.h строка 75

10.5.4.2 CC

const std::vector<Point2D>& Boundary::CC [inherited]

Константная ссылка на вектор из начал (и концов) панелей

См. определение в файле Boundary.h строка 78

10.5.4.3 numberInPassport

const size_t Boundary::numberInPassport [protected], [inherited]

Номер профиля в паспорте

См. определение в файле Boundary.h строка 69

10.5.4.4 oldSheets

Sheet Boundary::oldSheets [inherited]

Слои на профиле с предыдущего шага

См. определение в файле Boundary.h строка 104

10.5.4.5 sheetDim

size_t Boundary::sheetDim [inherited]

Размерность параметров каждого из слоев на каждой из панелей

Указывает, сколькими числами задается интенсивность каждого из слоев на каждой панели:

- 1 — одно число — задается только среднее значение;
- 2 — два числа — задается среднее значение и "наклон".

См. определение в файле Boundary.h строка 98

10.5.4.6 sheets

Sheet Boundary::sheets [inherited]

Слои на профиле

См. определение в файле Boundary.h строка 101

10.5.4.7 virtualWake

VirtualWake Boundary::virtualWake [inherited]

Виртуальный вихревой след конкретного профиля

См. определение в файле Boundary.h строка 84

10.5.4.8 vortexPerPanel

int Boundary::vortexPerPanel [inherited]

Число вихрей, рождаемых на каждой панели профиля и формирующих виртуальный вихревой след

См. определение в файле Boundary.h строка 81

10.5.4.9 W

const World2D& Boundary::W [protected], [inherited]

Константная ссылка на решаемую задачу

См. определение в файле Boundary.h строка 66

Объявления и описания членов классов находятся в файлах:

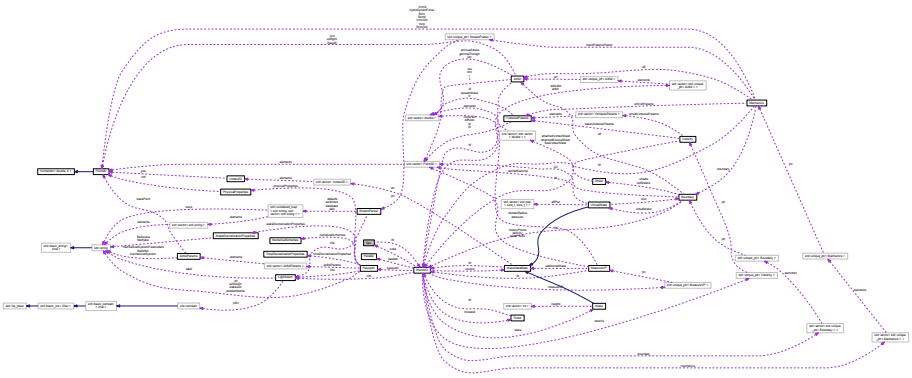
- Boundary/BoundaryConstLayerAver.h
- Boundary/BoundaryConstLayerAver.cpp

10.6 Класс Gpu

Класс, обеспечивающий возможность выполнения вычислений на GPU по технологии Nvidia CUDA.

#include <Gpu.h>

Граф связей класса Gpu:



Открытые члены

- Gpu (const World2D &W_)

  Конструктор
- ~Gpu ()
- void setAccelCoeff (double cft_)

  Установка коэффициента разгона потока
- void setCollapseCoeff (double pos_, double refLength_)

  Установка правой границы самого правого профиля (для организации увеличения радиуса коллапса)

Закрытые данные

- const World2D & W

  Константная ссылка на решаемую задачу

10.6.1 Подробное описание

Класс, обеспечивающий возможность выполнения вычислений на GPU по технологии Nvidia CUDA.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Gpu.h строка 59

10.6.2 Конструктор(ы)

10.6.2.1 Gpu()

```
Gpu::Gpu (
            const World2D & W_ )
```

Конструктор

Аргументы

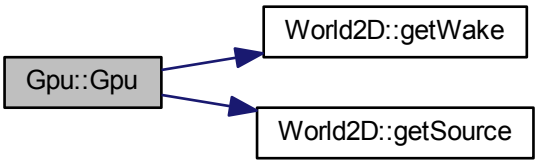
in	W↩ _	константная ссылка на решаемую задачу

См. определение в файле Gpu.cpp строка 42

```
43     : W(W_)
44 {
45 #if defined(__CUDACC__) || defined(USE_CUDA)
46
47 //   cuDevice(W.getParallel().myidWork);
```

```
48 //  cuDevice(W.getPassport().problemNumber);
49
50    cuSetConstants(sizeof(Vortex2D)/sizeof(double), Vortex2D::offsPos / sizeof(
   double), Vortex2D::offsGam / sizeof(double) );
51
52 /*
53    W.getWake().devVtxPtr = ReserveDevMem<double, sizeof(Vortex2D) / sizeof(double)>(INC_VORT_DEV,
   W.getWake().devNWake);
54
55    W.getWake().devVelPtr = ReserveDevMem<double, 2>(INC_VORT_DEV, n_CUDA_wake);
56    W.getWake().devRadPtr = ReserveDevMem<double, 1>(INC_VORT_DEV, n_CUDA_wake);
57    W.getWake().devI0Ptr = ReserveDevMem<double, 1>(INC_VORT_DEV, n_CUDA_wake);
58    W.getWake().devI1Ptr = ReserveDevMem<double, 1>(INC_VORT_DEV, n_CUDA_wake);
59    W.getWake().devI2Ptr = ReserveDevMem<double, 2>(INC_VORT_DEV, n_CUDA_wake);
60    W.getWake().devI3Ptr = ReserveDevMem<double, 2>(INC_VORT_DEV, n_CUDA_wake);
61
62
63    W.getWake().devMeshPtr = ReserveDevMem<int, 2>(INC_VORT_DEV, n_CUDA_wake);
64    W.getWake().devNeiPtr = ReserveDevMem<int, 1>(INC_VORT_DEV, n_CUDA_wake);
65
66    W.getWake().tmpVel.resize(n_CUDA_wake, { 0.0, 0.0 });
67    W.getWake().tmpRad.resize(n_CUDA_wake, 1.0e+5);
68
69    W.getWake().tmpI0.resize(n_CUDA_wake, 0.0);
70    W.getWake().tmpI1.resize(n_CUDA_wake, 0.0);
71    W.getWake().tmpI2.resize(n_CUDA_wake, { 0.0, 0.0 });
72    W.getWake().tmpI3.resize(n_CUDA_wake, { 0.0, 0.0 });
73    W.getWake().tmpNei.resize(n_CUDA_wake, 0);
74    //mesh.resize(n_CUDA_vel, { 0, 0 });
75 */
76
77    W.getWake().devNWake = 0;
78    W.getSource().devNWake = 0;
79    n_CUDA_wake = 0;
80    n_CUDA_afls = 0;
81 #endif
82 }
```

Граф вызовов:



10.6.2.2 ~Gpu()

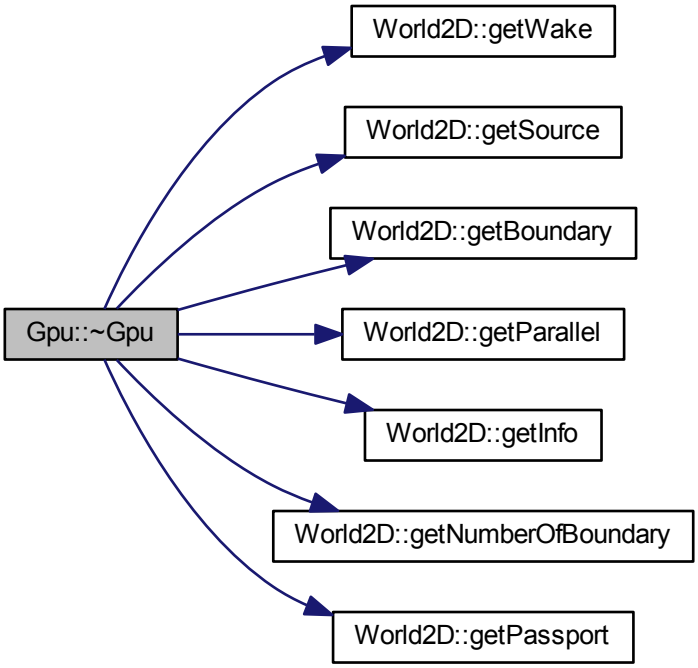
Gpu::~Gpu ( )

См. определение в файле Gpu.cpp строка 85

```
86 {
87 #if defined(__CUDACC__) || defined(USE_CUDA)
88    cuDeleteFromDev(W.getWake().devVtxPtr);
89    cuDeleteFromDev(W.getWake().devVelPtr);
90    cuDeleteFromDev(W.getWake().devRadPtr);
91    cuDeleteFromDev(W.getWake().devI0Ptr);
92    cuDeleteFromDev(W.getWake().devI1Ptr);
```

```
    cuDeleteFromDev(W.getWake().devI2Ptr);
    cuDeleteFromDev(W.getWake().devI3Ptr);
    cuDeleteFromDev(W.getWake().devMeshPtr);
    cuDeleteFromDev(W.getWake().devNeiPtr);

    if (W.getSource().vtx.size() > 0)
        cuDeleteFromDev(W.getSource().devVtxPtr);

    for (size_t s = 0; s < n_CUDA_afls; ++s)
    {
        cuDeleteFromDev(W.getBoundary(s).virtualWake.devVtxPtr);
        cuDeleteFromDev(W.getBoundary(s).virtualWake.devVelPtr);
        cuDeleteFromDev(W.getBoundary(s).virtualWake.devRadPtr);
        cuDeleteFromDev(W.getBoundary(s).virtualWake.devI0Ptr);
        cuDeleteFromDev(W.getBoundary(s).virtualWake.devI1Ptr);
        cuDeleteFromDev(W.getBoundary(s).virtualWake.devI2Ptr);
        cuDeleteFromDev(W.getBoundary(s).virtualWake.devI3Ptr);

        cuDeleteFromDev(W.getBoundary(s).afl.devRPtr);
        cuDeleteFromDev(W.getBoundary(s).afl.devRhsPtr);

        cuDeleteFromDev(W.getBoundary(s).afl.devFreeVortexSheetPtr);
        cuDeleteFromDev(W.getBoundary(s).afl.devAttachedVortexSheetPtr);
        cuDeleteFromDev(W.getBoundary(s).afl.devAttachedSourceSheetPtr);
    }

    if (n_CUDA_afls)
    {
        cuDeleteFromDev(dev_ptr_nPanels);
        cuDeleteFromDev(dev_ptr_nVortices);

        cuDeleteFromDev(dev_ptr_ptr_vtx);
        cuDeleteFromDev(dev_ptr_ptr_vel);
        cuDeleteFromDev(dev_ptr_ptr_rad);
        cuDeleteFromDev(dev_ptr_ptr_i0);
        cuDeleteFromDev(dev_ptr_ptr_i1);
        cuDeleteFromDev(dev_ptr_ptr_i2);
        cuDeleteFromDev(dev_ptr_ptr_i3);

        cuDeleteFromDev(dev_ptr_ptr_r);
        cuDeleteFromDev(dev_ptr_ptr_rhs);

        cuDeleteFromDev(dev_ptr_ptr_freeVortexSheet);
        cuDeleteFromDev(dev_ptr_ptr_attachedVortexSheet);
        cuDeleteFromDev(dev_ptr_ptr_attachedSourceSheet);
    }
#endif
}
```

Граф вызовов:



10.6.3 Методы

10.6.3.1 setAccelCoeff()

void Gpu::setAccelCoeff (
double cft_) [inline]

Установка коэффициента разгона потока

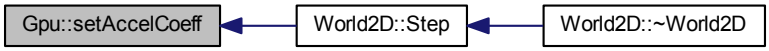
Аргументы

in	cft↩ _	множитель, соответствующий степени разгона потока

См. определение в файле Gpu.h строка 195

```
196     {
197 #if defined(__CUDACC__) || defined(USE_CUDA)
198         cuSetAccelCoeff(cft_);
199 #endif
200     }
```

Граф вызова функции:



10.6.3.2 setCollapseCoeff()

```
void Gpu::setCollapseCoeff (
            double pos_,
            double refLength_ )   [inline]
```

Установка правой границы самого правого профиля (для организации увеличения радиуса коллапса)

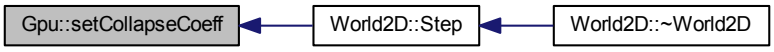
Аргументы

in	pos_	абсцисса правой границы самого правого профиля
in	ref↩ Length_	характерная длина, на которой происходит увеличение радиуса коллапса

См. определение в файле Gpu.h строка 207

```
208     {
209 #if defined(__CUDACC__) || defined(USE_CUDA)
210         cuSetCollapseCoeff(pos_, refLength_);
211 #endif
212     }
```

Граф вызова функции:



10.6.4 Данные класса

10.6.4.1 W

const World2D& Gpu::W [private]

Константная ссылка на решаемую задачу

См. определение в файле Gpu.h строка 63

Объявления и описания членов классов находятся в файлах:

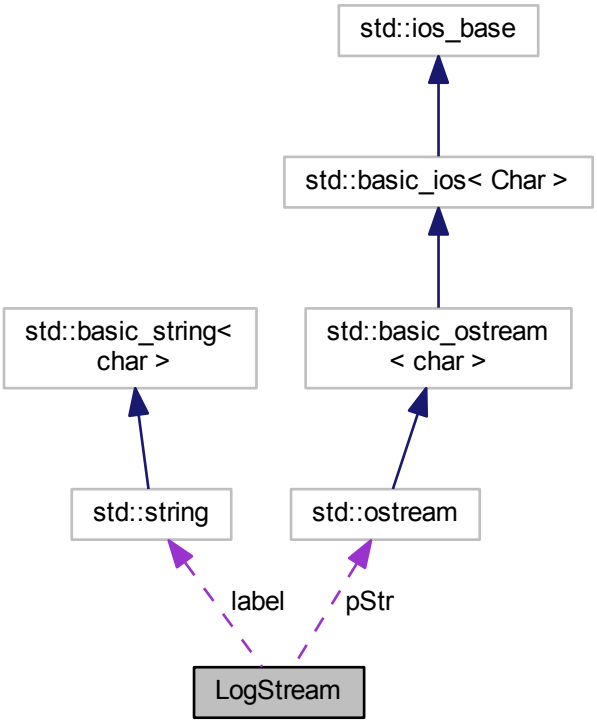
- Gpu/Gpu.h
- Gpu/Gpu.cpp

10.7 Класс LogStream

Класс, определяющий работу с потоком логов

#include <LogStream.h>

Граф связей класса LogStream:



Открытые члены

- LogStream ()

  Конструктор
- void assignStream (std::ostream *pStr_, const std::string &label_)

  Связывание потока логов с потоком вывода
- void inheritStream (LogStream &infoStream_, const std::string &label_)

  Связывание потока логов с потоком вывода от другого потока логов
- ~LogStream ()

  Деструктор
- void endl ()

  Вывод в поток логов пустой строки
- std::ostream & operator() (char c)

  Оператор вывода в поток логов

Закрытые данные

- std::ostream * pStr

  Указатель на поток вывода
- std::string label

  Метка потока

10.7.1 Подробное описание

Класс, определяющий работу с потоком логов

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле LogStream.h строка 56

10.7.2 Конструктор(ы)

10.7.2.1 LogStream()

LogStream::LogStream ( ) [inline]

Конструктор

Производит инициализацию потока логов пустым указателем

См. определение в файле LogStream.h строка 70

```
71        : pStr(nullptr), label("") {};
```

10.7.2.2 ~LogStream()

LogStream::~LogStream ( ) [inline]

Деструктор

См. определение в файле LogStream.h строка 99

```
99 {};
```

10.7.3 Методы

10.7.3.1 assignStream()

void LogStream::assignStream (
std::ostream * pStr_,
const std::string & label_) [inline]

Связывание потока логов с потоком вывода

Устанавливает связь потока логов с каким-либо потоком вывода

Аргументы

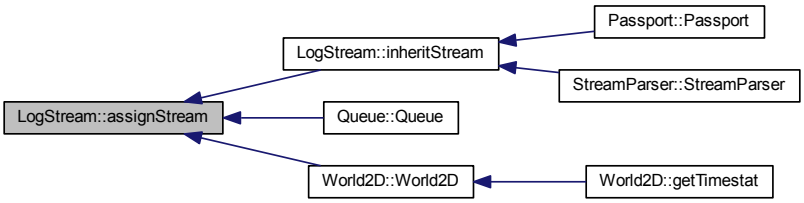
in	pStr↩ _	указатель на поток вывода
in	label↩ _	константная ссылка на метку потока

См. определение в файле LogStream.h строка 80

```
81    {
```

```
        pStr = pStr_;
        label = label_;
    }
```

Граф вызова функции:



10.7.3.2 endl()

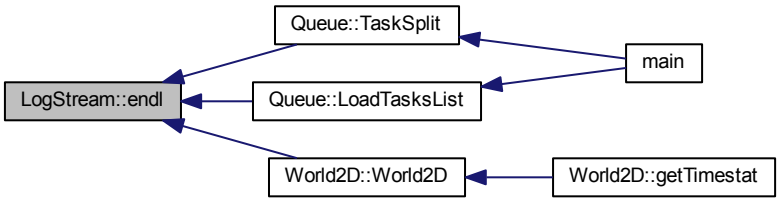
void LogStream::endl ( ) [inline]

Вывод в поток логов пустой строки

См. определение в файле LogStream.h строка 103

```
    {
        *pStr << std::endl;
    }
```

Граф вызова функции:



10.7.3.3 inheritStream()

void LogStream::inheritStream (
LogStream & infoStream_,
const std::string & label_) [inline]

Связывание потока логов с потоком вывода от другого потока логов

Устанавливает связь потока логов с потоком вывода, связанным с другим потоком логов

Аргументы

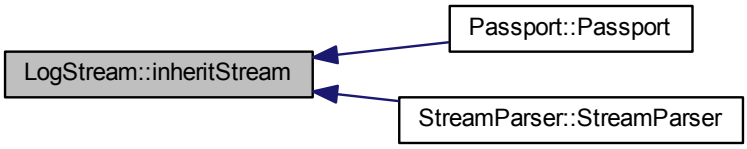
in	info↩ Stream_	существующий поток логов
in	label_	константная ссылка на дополнительную метку потока

См. определение в файле LogStream.h строка 93

```
94    {
95        assignStream(infoStream_.pStr, infoStream_.label + "->" + label_);
96    }
```

Граф вызовов:

Граф вызова функции:



10.7.3.4 operator()()

```
std::ostream& LogStream::operator() (
                char c )   [inline]
```

Оператор вывода в поток логов

Оператор (), возврящающий ссылку на поток вывода и печатающий соответствующую метку в зависимости от признака Типы признаков:

- i — признак вывода информации, метка info
- e — признак вывода ошибки, метка ERROR
- t — признак вывода телеметрии, метка tele
- - — признак вывода без метки (два пробела в начале строки)
- _ — признак вывода без метки (четыре пробела в начале строки)

Аргументы

in	c	признак типа выводимой информации

Возвращает

ссылку на поток вывода

См. определение в файле LogStream.h строка 121

```
122 {
123     if (pStr != nullptr)
124     {
125         switch (c)
126         {
127         case 'i':
128             *pStr << label << (label.length() > 0 ? " " : "") << "info: ";
129             break;
130
131         case 'e':
132             *pStr << label << (label.length() > 0 ? " " : "") << "ERROR: ";
133             break;
134
135         case 't':
136             *pStr << label << (label.length() > 0 ? " " : "") << "tele: ";
137             break;
138
139         case '-':
140             *pStr << "  ";
141             break;
142
143         case '_':
144             *pStr << "    ";
145             break;
146         }
147         return *pStr;
148     }
149     else
150     {
151         pStr = new std::ostream(NULL);
152         return *pStr;
153     }
154 }
```

10.7.4 Данные класса

10.7.4.1 label

std::string LogStream::label [private]

Метка потока

См. определение в файле LogStream.h строка 63

10.7.4.2 pStr

std::ostream* LogStream::pStr [private]

Указатель на поток вывода

См. определение в файле LogStream.h строка 60

Объявления и описания членов класса находятся в файле:

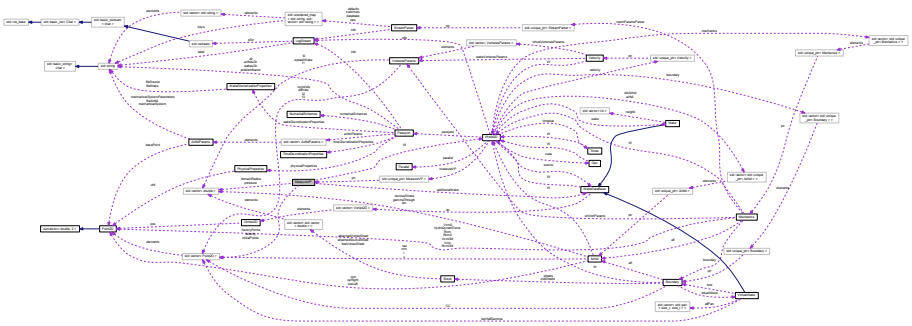
- LogStream/LogStream.h

10.8 Класс MeasureVP

Класс, отвечающий за вычисление поля скорости и давления в заданых точках для вывода

#include <MeasureVP.h>

Граф связей класса MeasureVP:



Открытые члены

- MeasureVP (const World2D &W_)

  Конструктор
- ~MeasureVP ()

  Деструктор
- void ReadPointsFromFile (const std::string &dir)

  Чтение точек, в которых нужно посчитать давление и скорость
- void CalcSaveVP (const std::string &dir, size_t step, timePeriod &time)

  Расчет и сохранение в файл поля скоростей и давления

Защищенные данные

- const World2D & W

  Константная ссылка на решаемую задачу

Закрытые данные

- std::vector< Point2D > initialPoints

  Точки, которые считываются из файла (давление пишется в vtk-файлы)
- std::vector< Point2D > historyPoints

  Точки, которые считываются из файла (давление пишется в vtk и csv-файлы)
- WakeDataBase additionalWake

  Точки, в которых нужно вычислять в данный момент времени Хранятся в виде "мнимой" системы вихрей, чтобы воспользоваться методом CalcConvVeloToSetOfPoints(...)
- std::vector< Point2D > velocity

  Скорости в нужных точках
- std::vector< double > domainRadius

  Радиусы вихрей в нужных точках
- std::vector< double > pressure

  Давление в нужных точках

10.8.1 Подробное описание

Класс, отвечающий за вычисление поля скорости и давления в заданых точках для вывода

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле MeasureVP.h строка 66

10.8.2 Конструктор(ы)

10.8.2.1 MeasureVP()

```
MeasureVP::MeasureVP (
            const World2D & W_ )   [inline]
```

Конструктор

Аргументы

in	W↩ _	константная ссылка на решаемую задачу

См. определение в файле MeasureVP.h строка 99

```
99 : W(W_), additionalWake(W_) {};
```

10.8.2.2 ∼MeasureVP()

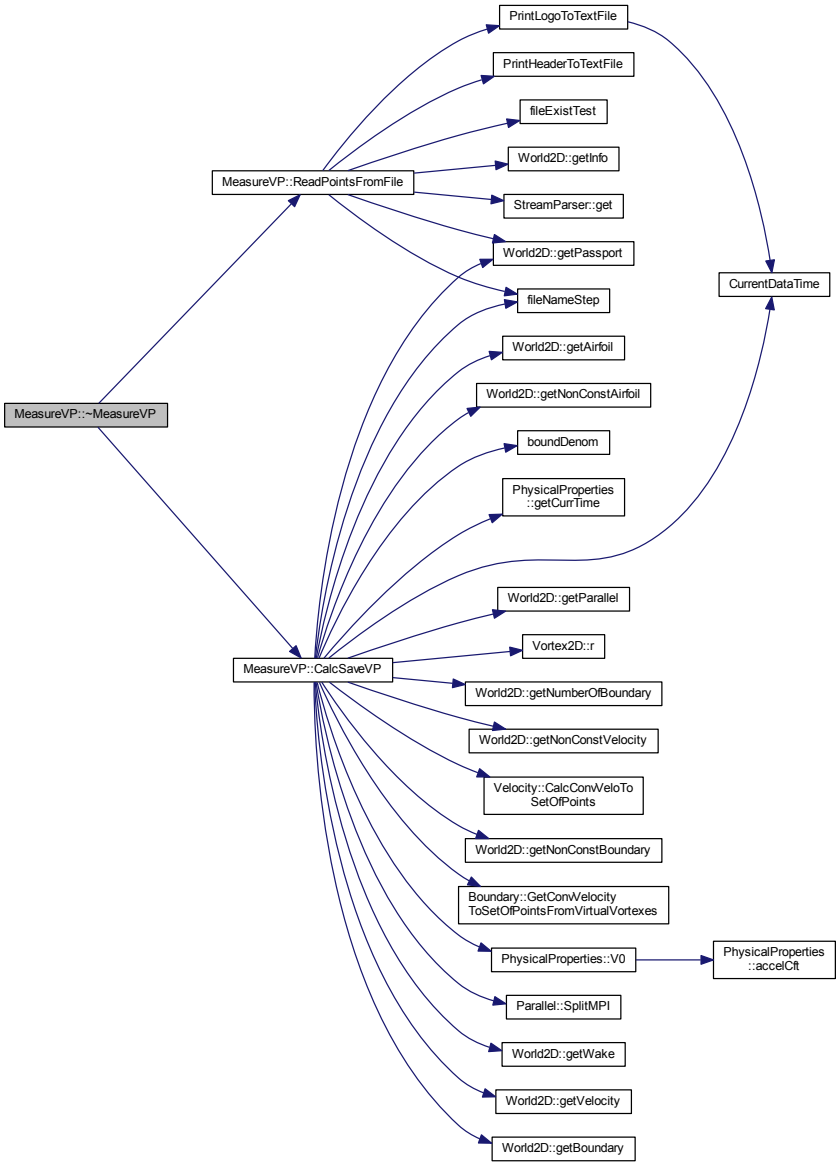
MeasureVP::∼MeasureVP ( ) [inline]

Деструктор

См. определение в файле MeasureVP.h строка 102

```
102 { };
```

Граф вызовов:



10.8.3 Методы

10.8.3.1 CalcSaveVP()

void MeasureVP::CalcSaveVP (
const std::string & dir,
size_t step,
timePeriod & time)

Расчет и сохранение в файл поля скоростей и давления

Аргументы

in	dir	константная ссылка на строку, задающую каталог, куда сохранять файл с вихревым следом
in	step	номер кадра для сохранения
out	time	ссылка на промежуток времени — пару чисел (время начала и время конца операции)

Необходимо сделать Для успокоения добавить WakeSyncronize(), но после обучения его проверке hash

Необходимо сделать Пока передаем только средние значения свободного слоя на панелях

См. определение в файле MeasureVP.cpp строка 110

```
111 {
112     time.first = omp_get_wtime();
113     std::vector<Point2D> velConvWake;
114     std::vector<std::vector<Point2D>> velConvBou;
115
116     if (W.getParallel().myidWork == 0)
117     {
118         additionalWake.vtx.clear();
119         additionalWake.vtx.resize(0);
120
121         Vortex2D addvtx;
122
123         velocity.clear();
124         velocity.resize(0);
125
126         pressure.clear();
127         pressure.resize(0);
128
129         domainRadius.clear();
130         domainRadius.resize(0);
131
132         //  std::vector<WakeDataBase> additionalWakeSource; //дополнительная база данных вихрей на местах
    источников для расчета скоростей
133         //  additionalWakeSource.resize(W.getNumberOfBoundary());
134
135         //  std::vector<std::vector<Point2D>> additionalWakeSourceConvVelo;
136         //  std::vector<std::vector<Point2D>> additionalWakeSourceDiffVelo;
137
138
139         //отключаем проверку того, что точки расположены вне профиля
140         /*
141         //определяем точки, которые не находятся внутри профилей
142         bool inside;
143         for (size_t i = 0; i < initialPoints.size(); i++)
144         {
145             inside = false;
146             for (size_t j = 0; j < W.getNumberOfAirfoil(); j++)
147             {
148                 if (W.getAirfoil(j).isInsideGabarits(initialPoints[i]) &&
    W.getAirfoil(j).IsPointInAirfoil(initialPoints[i]))
149                 {
150                     inside = true;
151                     break;
152                 }
153             }
154             if (!inside)
155             {
156                 addvtx.r() = initialPoints[i];
157                 additionalWake.vtx.push_back(addvtx);
158             }
159         }
160         */
161
162         //вычисляем скорости и давления во всех точках, а не только в тех, которые вне профиля
163         for (size_t i = 0; i < initialPoints.size(); i++)
164         {
165             addvtx.r() = initialPoints[i];
166             additionalWake.vtx.push_back(addvtx);
```

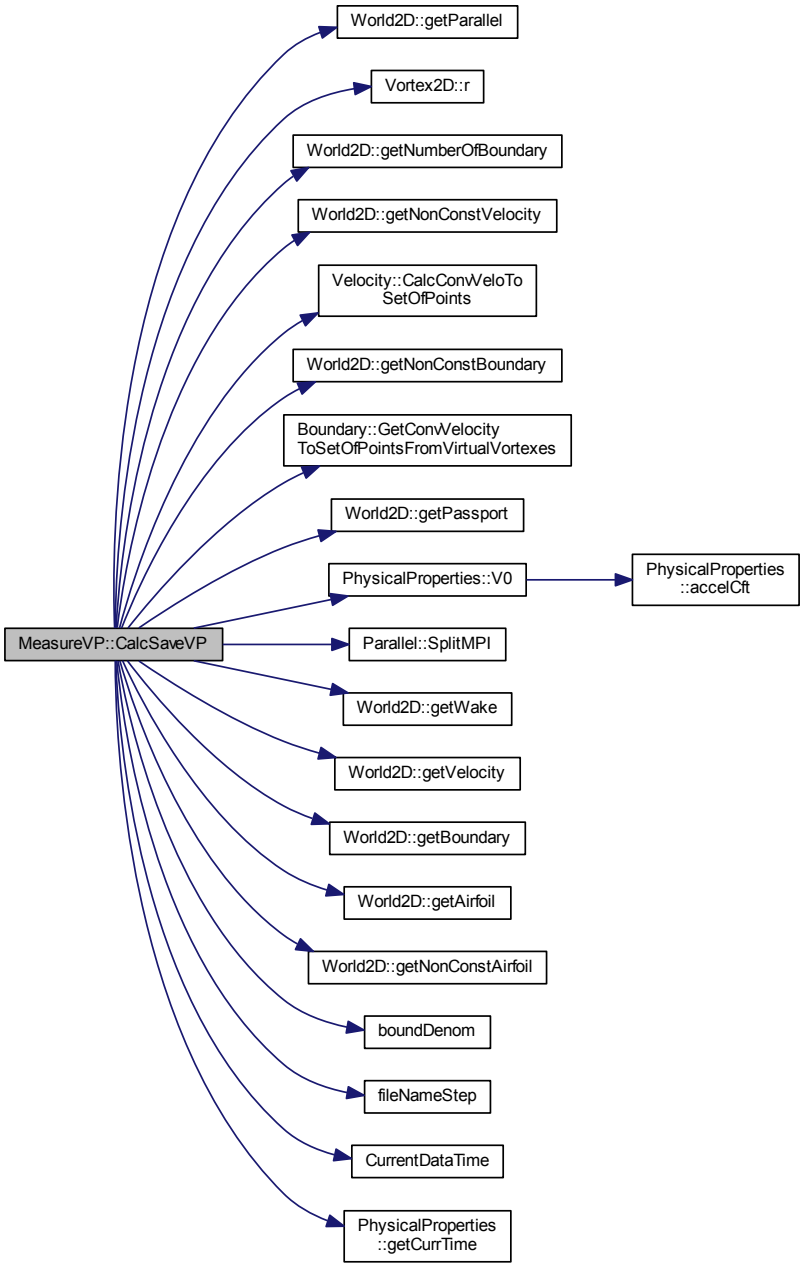
```
        }

        for (size_t i = 0; i < historyPoints.size(); i++)
        {
            addvtx.r() = historyPoints[i];
            additionalWake.vtx.push_back(addvtx);
        }

        velConvWake.resize(additionalWake.vtx.size(), { 0.0, 0.0 });
    }

    int addWSize = (int)additionalWake.vtx.size();
    MPI_Bcast(&addWSize, 1, MPI_INT, 0, W.getParallel().commWork);

    velConvBou.resize(W.getNumberOfBoundary());
    for (size_t i = 0; i < W.getNumberOfBoundary(); i++)
        velConvBou[i].resize(additionalWake.vtx.size(), { 0.0, 0.0 });

    /*#if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_CONV_TOWAKE))
        {
        VelocityClass.GPUCalcConvVeloToSetOfPoints(additionalWake, velConvWake, domainRadius);
        for (size_t i = 0; i < W.getNumberOfBoundary(); i++)
        {
        Boundary& BoundaryClass = W.getNonConstBoundary(i);
        BoundaryClass.GPUGetConvVelocityToSetOfPointsFromVirtualVortexes(additionalWake, velConvBou[i]);
        }
        }
        #else
        */
    {

        W.getNonConstVelocity().CalcConvVeloToSetOfPoints(additionalWake, velConvWake, domainRadius);

        for (size_t i = 0; i < W.getNumberOfBoundary(); i++)
        {
            W.getNonConstBoundary(i).GetConvVelocityToSetOfPointsFromVirtualVortexes(additionalWake, velConvBou[i]);
        }
    }
//#endif


    if (W.getParallel().myidWork == 0)
    {
        velocity.assign(addWSize, W.getPassport().physicalProperties.V0());
        pressure.resize(addWSize);

        for (size_t i = 0; i < additionalWake.vtx.size(); i++)
            velocity[i] += velConvWake[i];

        for (size_t bou = 0; bou < velConvBou.size(); bou++)
        for (size_t j = 0; j < additionalWake.vtx.size(); j++)
            velocity[j] += velConvBou[bou][j];
    }

    int id = W.getParallel().myidWork;
    parProp par = W.getParallel().SplitMPI(addWSize);


    std::vector<Point2D> locVelocity;
    locVelocity.resize(par.myLen);
    MPI_Scatterv(velocity.data(), par.len.data(), par.disp.data(), Point2D::mpiPoint2D, locVelocity.data(), par.myLen, Point2D::mpiPoint2D, 0, W.getParallel().commWork);

    std::vector<double> locPressure;
    locPressure.resize(par.myLen, 0.0);

    WakeDataBase locAdditionalWake(W);
    locAdditionalWake.vtx.resize(par.myLen);
    MPI_Scatterv(additionalWake.vtx.data(), par.len.data(), par.disp.data(), Vortex2D::mpiVortex2D, locAdditionalWake.vtx.data(), par.myLen, Vortex2D::mpiVortex2D, 0, W.getParallel().commWork);


    W.getNonConstVelocity().wakeVortexesParams.convVelo.resize(W.getWake().vtx.size());
    MPI_Bcast(W.getNonConstVelocity().wakeVortexesParams.convVelo.data(), (int)(W.getVelocity().wakeVortexesParams.convVelo.size()), Point2D::mpiPoint2D, 0, W.getParallel().commWork);


```

```
    std::vector<std::vector<double>> gamAverPan;
    gamAverPan.resize(W.getNumberOfBoundary());

    if (W.getParallel().myidWork == 0)
    for (int bou = 0; bou < W.getNumberOfBoundary(); bou++)
    {
        gamAverPan[bou].resize(W.getBoundary(bou).sheets.freeVortexSheet.size());
        for (int j = 0; j < W.getBoundary(bou).sheets.freeVortexSheet.size(); ++j)
            gamAverPan[bou][j] = W.getBoundary(bou).sheets.freeVortexSheet[j][0];
    }

    for (int bou = 0; bou < W.getNumberOfBoundary(); bou++)
    {
        int sz = (int)(W.getBoundary(bou).sheets.freeVortexSheet.size());
        MPI_Bcast(&sz, 1, MPI_INT, 0, W.getParallel().commWork);
        if (id > 0)
            gamAverPan[bou].resize(sz);
        MPI_Bcast(gamAverPan[bou].data(), sz, MPI_DOUBLE, 0, W.getParallel().commWork);
    }

    for (int bou = 0; bou < W.getNumberOfBoundary(); bou++)
    {
        int sz = (int)(W.getAirfoil(bou).gammaThrough.size());
        MPI_Bcast(&sz, 1, MPI_INT, 0, W.getParallel().commWork);
        MPI_Bcast(W.getNonConstAirfoil(bou).gammaThrough.data(), sz, MPI_DOUBLE, 0, W.getParallel().commWork);
    }

    for (int bou = 0; bou < W.getNumberOfBoundary(); bou++)
    {
        int sz = (int)(W.getBoundary(bou).virtualWake.vecHalfGamma.size());
        MPI_Bcast(&sz, 1, MPI_INT, 0, W.getParallel().commWork);
        if (id > 0)
            W.getNonConstBoundary(bou).virtualWake.vecHalfGamma.resize(sz);
        MPI_Bcast(W.getNonConstBoundary(bou).virtualWake.vecHalfGamma.data(), sz, Point2D::mpiPoint2D, 0, W.getParallel().commWork);
    }

    Point2D dri;
    Point2D Vi;
    Point2D vi;


    const Point2D& V0 = W.getPassport().physicalProperties.V0();
    const double& eps2 = W.getPassport().wakeDiscretizationProperties.eps2;
    const double& dt = W.getPassport().timeDiscretizationProperties.dt;


    double P0 = /*1*/ 0.5 * V0 * V0;


    Point2D cPan;

#pragma warning (push)
#pragma warning (disable: 4101)
    double alpha;
    double dst2eps;
#pragma warning (pop)

#pragma omp parallel for default(none) private(alpha, dri, Vi, vi, dst2eps, cPan) shared(P0, dt, eps2, V0, gamAverPan, id, locAdditionalWake, locPressure, locVelocity, par, std::cout)
    for (int locI = 0; locI < par.myLen; ++locI)
    {
        //int i = locI + par.myDisp;
        locPressure[locI] = P0;
        locPressure[locI] -=  0.5 * (locVelocity[locI] * locVelocity[locI]); //2

        const Point2D& pt = locAdditionalWake.vtx[locI].r();

        for (int j = 0; j < W.getWake().vtx.size(); j++)
        {
            dri = pt - W.getWake().vtx[j].r();
            Vi = W.getVelocity().wakeVortexesParams.convVelo[j] + V0;

            dst2eps = boundDenom(dri.length2(), eps2);
```

```
            vi = IDPI * W.getWake().vtx[j].g() / dst2eps * dri.kcross();
            locPressure[locI] += vi * Vi; //3
        }

        for (int bou = 0; bou < W.getNumberOfBoundary(); bou++)
        for (int j = 0; j < W.getBoundary(bou).virtualWake.vtx.size(); j++)
        {
            dri = pt - W.getBoundary(bou).virtualWake.vtx[j].r();

            dst2eps = boundDenom(dri.length2(), eps2);

            Vi = W.getBoundary(bou).virtualWake.vecHalfGamma[j];
            vi = IDPI * W.getBoundary(bou).virtualWake.vtx[j].g() / dst2eps * dri.kcross();

            locPressure[locI] +=  vi * Vi; //4
        }

        for (int bou = 0; bou < W.getNumberOfBoundary(); bou++)
        {
            const Point2D& rcm = W.getAirfoil(bou).rcm;

            for (int j = 0; j < W.getAirfoil(bou).np; j++)
            {
                cPan = 0.5 * (W.getAirfoil(bou).r[j] + W.getAirfoil(bou).r[j + 1]);
                alpha = atan2((cPan - pt) ^ (rcm - pt),   (cPan - pt)*(rcm - pt));

                locPressure[locI] += IDPI * alpha * gamAverPan[bou][j] * W.getAirfoil(bou).len[j] / dt;

                locPressure[locI] -= IDPI * alpha * W.getAirfoil(bou).gammaThrough[j] / dt;
            }
        }//for bou
    }//for i

    MPI_Gatherv(locPressure.data(), par.myLen, MPI_DOUBLE, pressure.data(), par.len.data(), par.disp.data(), MPI_DOUBLE, 0, W.getParallel().commWork);


    if (W.getParallel().myidWork == 0)
    {
        std::string fname = fileNameStep("VelPres", W.getPassport().timeDiscretizationProperties.nameLength, step, "vtk");

        std::ofstream outfile;

        outfile.open(dir + "velPres/" + fname);

        outfile << "# vtk DataFile Version 2.0" << std::endl;
        outfile << "VM2D VTK result: " << (dir + "velPres/" + fname).c_str() << " saved " << CurrentDataTime() << std::endl;
        outfile << "ASCII" << std::endl;
        outfile << "DATASET UNSTRUCTURED_GRID" << std::endl;
        outfile << "POINTS " << additionalWake.vtx.size() << " float" << std::endl;

        for (size_t i = 0; i < additionalWake.vtx.size(); i++)
        {
            double xi = (additionalWake.vtx[i].r())[0];
            double yi = (additionalWake.vtx[i].r())[1];
            outfile << xi << " " << yi << " " << "0.0" << std::endl;
        }//for i

        outfile << "CELLS " << additionalWake.vtx.size() << " " << 2 * additionalWake.vtx.size() << std::endl;
        for (size_t i = 0; i < additionalWake.vtx.size(); ++i)
            outfile << "1 " << i << std::endl;

        outfile << "CELL_TYPES " << additionalWake.vtx.size() << std::endl;
        for (size_t i = 0; i < additionalWake.vtx.size(); ++i)
            outfile << "1" << std::endl;

        outfile << std::endl;
        outfile << "POINT_DATA " << additionalWake.vtx.size() << std::endl;

        outfile << "VECTORS V float" << std::endl;
        for (size_t i = 0; i < additionalWake.vtx.size(); i++)
        {
            outfile << velocity[i][0] << " " << velocity[i][1] << " 0.0" << std::endl;
        }//for i

        outfile << std::endl;
```

```
391
392         outfile << "SCALARS P float 1" << std::endl;
393         outfile << "LOOKUP_TABLE default" << std::endl;
394
395         for (size_t i = 0; i < additionalWake.vtx.size(); i++)
396         {
397             outfile << pressure[i] << std::endl;
398         }//for i
399
400         outfile.close();
401
402         //Вывод в csv-файлы
403         for (int q = 0; q < historyPoints.size(); ++q)
404         {
405             std::string VPFileNameCsv;
406             VPFileNameCsv = W.getPassport().dir + "velPres/" +
    fileNameStep("VP-atPoint-", 2, q, "csv");
407
408             std::ofstream VPFileCsv(VPFileNameCsv.c_str(), std::ios::app);
409
410             VPFileCsv << W.getPassport().physicalProperties.
    getCurrTime() << ","
411                 << velocity[initialPoints.size() + q][0] << ","
412                 << velocity[initialPoints.size() + q][1] << ","
413                 << pressure[initialPoints.size() + q] << std::endl;
414             VPFileCsv.close();
415         }
416
417     }
418
419
420     time.second = omp_get_wtime();
421
422 }//CalcSaveVP()
```

Граф вызовов:



Граф вызова функции:

10.8.3.2 ReadPointsFromFile()

```
void MeasureVP::ReadPointsFromFile (
                const std::string & dir )
```

Чтение точек, в которых нужно посчитать давление и скорость

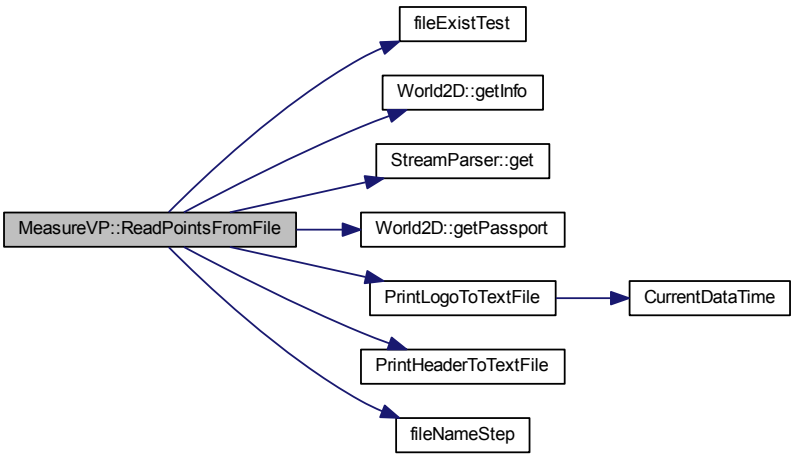
Аргументы

in	dir	константная ссылка на строку — имя каталога, где лежит считываемый файл

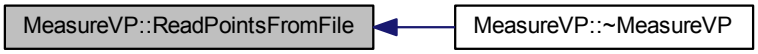
См. определение в файле MeasureVP.cpp строка 60

```
{
    std::string filename = dir + "pointsVP";
    std::ifstream VPFile;

    if (fileExistTest(filename, W.getInfo()))
    {
        std::stringstream VPFile(Preprocessor(filename).resultString);

        StreamParser VPParser(W.getInfo(), "velocity & pressure parser", VPFile);

        VPParser.get("points", initialPoints);
        VPParser.get("history", historyPoints);

        if (historyPoints.size() > 0)
        {
#if defined(_WIN32)
            _mkdir((dir + "velPres").c_str());
#else
            mkdir((dir + "velPres").c_str(), S_IRWXU | S_IRGRP | S_IROTH);
#endif

            std::string VPFileNameList = W.getPassport().dir + "velPres/listPoints";
            std::ofstream VPFileList(VPFileNameList.c_str());

            PrintLogoToTextFile(VPFileList, VPFileNameList.c_str(), "List of points, where velocity and pressure are measured in csv-files");

            PrintHeaderToTextFile(VPFileList, "No.     FileName     X     Y");

            for (int q = 0; q < historyPoints.size(); ++q)
            {
                VPFileList << '\n' << q << '\t' << fileNameStep("VP-atPoint-", 2, q, "csv") << '\t' << historyPoints[q][0] << '\t' << historyPoints[q][1];

                std::string VPFileNameCsv;
                VPFileNameCsv = W.getPassport().dir + "velPres/" + fileNameStep("VP-atPoint-", 2, q, "csv");

                std::ofstream VPFileCsv(VPFileNameCsv.c_str());

                VPFileCsv << "t,Vx,Vy,p" << std::endl;
                VPFileCsv.close();
                 VPFileCsv.clear();
            }

            VPFileList.close();
            VPFileList.clear();
        }
    }
}//ReadPointsFromFile(...)
```

Граф вызовов:



Граф вызова функции:



10.8.4 Данные класса

10.8.4.1 additionalWake

WakeDataBase MeasureVP::additionalWake [private]

Точки, в которых нужно вычислять в данный момент времени Хранятся в виде "мнимой" системы вихрей, чтобы воспользоваться методом CalcConvVeloToSetOfPoints(...)

См. определение в файле MeasureVP.h строка 77

10.8.4.2 domainRadius

std::vector<double> MeasureVP::domainRadius [private]

Радиусы вихрей в нужных точках

См. определение в файле MeasureVP.h строка 83

10.8.4.3 historyPoints

std::vector<Point2D> MeasureVP::historyPoints [private]

Точки, которые считываются из файла (давление пишется в vtk и csv-файлы)

См. определение в файле MeasureVP.h строка 73

10.8.4.4 initialPoints

std::vector<Point2D> MeasureVP::initialPoints [private]

Точки, которые считываются из файла (давление пишется в vtk-файлы)

См. определение в файле MeasureVP.h строка 70

10.8.4.5 pressure

std::vector<double> MeasureVP::pressure [private]

Давление в нужных точках

См. определение в файле MeasureVP.h строка 86

10.8.4.6 velocity

std::vector<Point2D> MeasureVP::velocity [private]

Скорости в нужных точках

См. определение в файле MeasureVP.h строка 80

10.8.4.7 W

const World2D& MeasureVP::W [protected]

Константная ссылка на решаемую задачу

Необходимо сделать Сделать учет давления на бесконечности

См. определение в файле MeasureVP.h строка 93

Объявления и описания членов классов находятся в файлах:

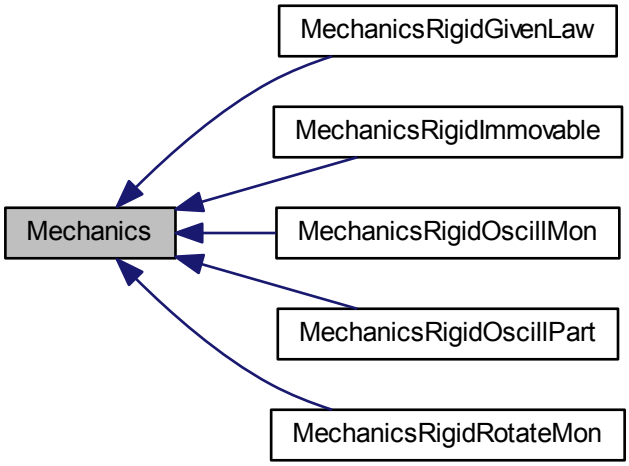
- MeasureVP/MeasureVP.h
- MeasureVP/MeasureVP.cpp

10.9 Класс Mechanics

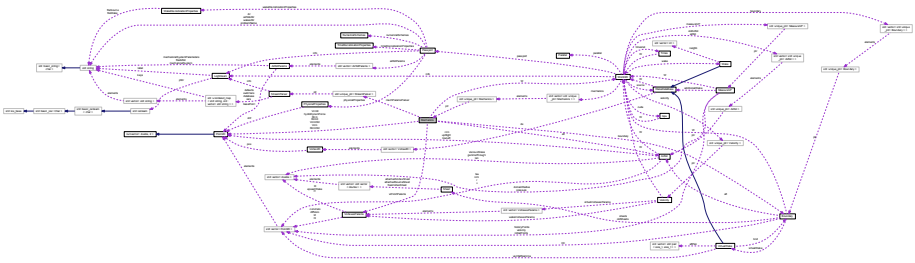
Абстрактный класс, определяющий вид механической системы

#include <Mechanics.h>

Граф наследования:Mechanics:



Граф связей класса Mechanics:



Открытые члены

- virtual void ReadSpecificParametersFromDictionary ()=0

  Чтение параметров конкретной механической системы
- Mechanics (const World2D &W_, size_t numberInPassport_, int degOfFreedom_, bool isMoves↩_, bool isDeform_, bool isRotate_, Point2D Vcm0_, Point2D Rcm0_, double Wcm0_, double Phi0_)

  Конструктор
- virtual ~Mechanics ()

  Деструктор
- virtual void GetHydroDynamForce (timePeriod &time)=0

Вычисление гидродинамической силы, действующей на профиль

- void GenerateForcesHeader ()

  Генерация заголовка файла нагрузок
- void GeneratePositionHeader ()

  Генерация заголовка файла положения профиля
- void GenerateForcesString ()

  Сохранение строки со статистикой в файл нагрузок
- void GeneratePositionString ()

  Сохранение строки со статистикой в файл нагрузок
- virtual Point2D VeloOfAirfoilRcm (double currTime)=0

  Вычисление скорости центра масс профиля
- virtual Point2D PositionOfAirfoilRcm (double currTime)=0

  Вычисление положения центра масс профиля
- virtual void VeloOfAirfoilPanels (double currTime)=0

  Вычисление скоростей начал панелей
- virtual void FillMechanicsRowsAndCross (Eigen::MatrixXd &row, Eigen::MatrixXd &cross)=0

  Заполнение строк в матрице, соответствующих механической системе. А также пересечений строк и столбцов, соответствующих мех. системе
- virtual void FillMechanicsRhs (std::vector< double > &rhs)=0

  Заполнение правых частей, соответствующих механической системе
- virtual void FillAtt (Eigen::MatrixXd &row, Eigen::MatrixXd &rhs)=0

  Заполнение правых частей (расщепленная схема) либо коэффициентов в матрице при скорости (монолитный подход), соответствующих присоединенным слоям
- virtual void SolutionToMechanicalSystem (Eigen::VectorXd &col)=0

  Извлечение из решения параметров механической системы
- virtual void Move ()=0

  Перемещение профиля в соответствии с законом

Открытые атрибуты

- const bool isMoves

  Переменная, отвечающая за то, двигается профиль или нет
- const bool isDeform

  Переменная, отвечающая за то, деформируется профиль или нет
- const bool isRotate

  Переменная, отвечающая за то, может профиль вращаться или нет
- const size_t degOfFreedom

  Количество степеней свободы
- Point2D hydroDynamForce

  Вектор гидродинамической силы и момент, действующие на профиль
- double hydroDynamMoment

Защищенные данные

- const World2D & W

  Константная ссылка на решаемую задачу
- const size_t numberInPassport

  Номер профиля в паспорте
- Airfoil & afl
- const Boundary & boundary

- const VortexesParams & virtVortParams
- std::unique_ptr< StreamParser > mechParamsParser

  Умный указатель на парсер параметров механической системы
- const Point2D Vcm0

  Начальная скорость центра и угловая скорость
- const double Wcm0
- const Point2D Rcm0

  Начальное положение профиля
- double Phi0
- Point2D Vcm

  Текущие скорость центра и угловая скорость
- double Wcm
- Point2D Rcm

  Текущие положение профиля
- double Phi
- Point2D VcmOld

  Скорость и отклонение с предыдущего шага
- double WcmOld
- Point2D RcmOld

  Текущие положение профиля
- double PhiOld

Закрытые члены

- void ReadParametersFromDictionary ()

  Парсинг списка параметров механической системы

10.9.1 Подробное описание

Абстрактный класс, определяющий вид механической системы

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Mechanics.h строка 67

10.9.2 Конструктор(ы)

10.9.2.1 Mechanics()

```
Mechanics::Mechanics (
            const World2D & W_,
            size_t numberInPassport_,
            int degOfFreedom_,
            bool isMoves_,
            bool isDeform_,
            bool isRotate_,
            Point2D Vcm0_,
            Point2D Rcm0_,
            double Wcm0_,
            double Phi0_ )
```

Конструктор

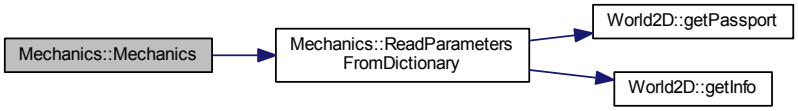
Аргументы

in	W_	константная ссылка на решаемую задачу
in	numberIn↩ Passport_	номер профиля в паспорте задачи
in	degOfFreedom_	количество степеней свободы
in	isMoves_	является ли профиль подвижным (1 - является, 0 - не является)
in	isDeform_	является ли профиль деформируемым (1 - является, 0 - не является)
in	isRotate_	является ли профиль вращающимся (1 - является, 0 - не является)
in	Vcm0_	- скорость центра масс
in	Rcm0_	- положение центра масс
in	Wcm0_	- угловая скорость центра масс
in	Phi0_	- угол поворота центра масс

См. определение в файле Mechanics.cpp строка 43

```
44 : W(W_), numberInPassport(numberInPassport_), afl(W_.
     getNonConstAirfoil(numberInPassport_)), boundary(W_.
     getBoundary(numberInPassport_)), virtVortParams(W_.
     getVelocity().virtualVortexesParams[numberInPassport_]),
     degOfFreedom(degOfFreedom_), isMoves(isMoves_), isDeform(isDeform_),
     isRotate(isRotate_),\
45 Vcm0(Vcm0_), Rcm0(Rcm0_), Wcm0(Wcm0_), Phi0(Phi0_)
46 {
47     Vcm = Vcm0;
48     Wcm = Wcm0;
49     Rcm = Rcm0;
50     Phi = Phi0;
51     VcmOld = Vcm0;
52     WcmOld = Wcm0;
53     RcmOld = Rcm0;
54     PhiOld = Phi0;
55     ReadParametersFromDictionary();
56 };
```

Граф вызовов:



10.9.2.2 ~Mechanics()

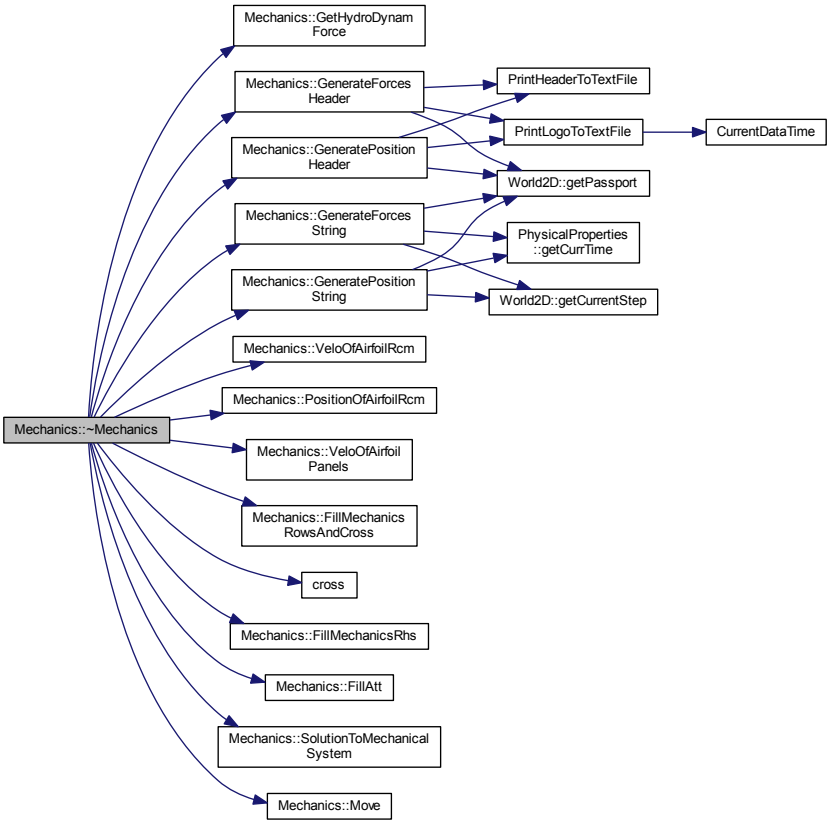
virtual Mechanics::~Mechanics ( ) [inline], [virtual]

Деструктор

См. определение в файле Mechanics.h строка 151

```
151 { };
```

Граф вызовов:



10.9.3 Методы

10.9.3.1 FillAtt()

```
virtual void Mechanics::FillAtt (
            Eigen::MatrixXd & row,
            Eigen::MatrixXd & rhs )   [pure virtual]
```

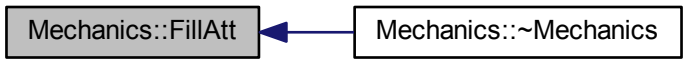
Заполнение правых частей (расщепленная схема) либо коэффициентов в матрице при скорости (монолитный подход), соответствующих присоединенным слоям

Аргументы

in	row	строки (количество строк = количество степеней свободы, количество элементов в строках = количество неизвестных в локальной матрице)
in	rhs	вектор правой части для механики одного профиля (количество элементов = количеству степеней свободы)

Замещается в MechanicsRigidRotateMon, MechanicsRigidOscillPart, MechanicsRigidOscillMon, MechanicsRigidGivenLaw и MechanicsRigidImmovable.

Граф вызова функции:



10.9.3.2 FillMechanicsRhs()

```
virtual void Mechanics::FillMechanicsRhs (
            std::vector< double > & rhs )   [pure virtual]
```

Заполнение правых частей, соответствующих механической системе

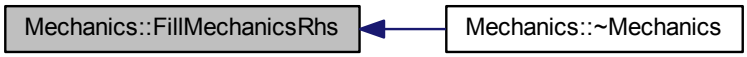
Аргументы

in	rhs	вектор правой части для механики одного профиля (количество элементов = количеству степеней свободы)

Замещается в MechanicsRigidRotateMon, MechanicsRigidOscillPart, MechanicsRigidOscillMon,

MechanicsRigidGivenLaw и MechanicsRigidImmovable.

Граф вызова функции:



10.9.3.3 FillMechanicsRowsAndCross()

```
virtual void Mechanics::FillMechanicsRowsAndCross (
            Eigen::MatrixXd & row,
            Eigen::MatrixXd & cross )   [pure virtual]
```

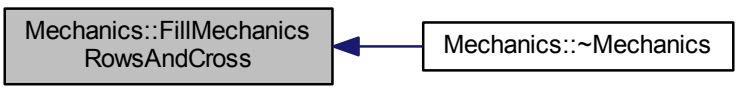
Заполнение строк в матрице, соответствующих механической системе. А также пересечений строк и столбцов, соответствующих мех. системе

Аргументы

in	row	строки (количество строк = количество степеней свободы, количество элементов в строках = количество неизвестных в локальной матрице)
in	cross	пересечения (количество столбцов = количество степеней свободы, количество элементов в столбцах = количество степеней свободы)

Замещается в MechanicsRigidRotateMon, MechanicsRigidOscillPart, MechanicsRigidOscillMon, MechanicsRigidGivenLaw и MechanicsRigidImmovable.

Граф вызова функции:



10.9.3.4 GenerateForcesHeader()

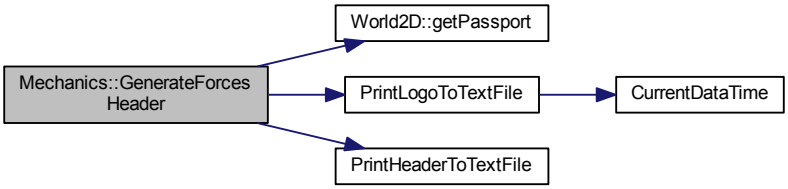
void Mechanics::GenerateForcesHeader ( )

Генерация заголовка файла нагрузок

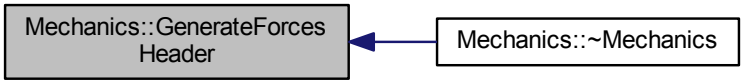
См. определение в файле Mechanics.cpp строка 69

```
70 {
71     std::stringstream forceFileName, forceFileNameCsv;
72     forceFileName << W.getPassport().dir << "forces-airfoil-" <<
    numberInPassport;
73     forceFileNameCsv << W.getPassport().dir << "forces-airfoil-" <<
    numberInPassport << ".csv";
74
75     std::ofstream newForcesFile(forceFileName.str());
76     std::ofstream newForcesFileCsv(forceFileNameCsv.str());
77
78     PrintLogoToTextFile(newForcesFile, forceFileName.str(), "Hydrodynamic loads for the
     airfoil " + W.getPassport().airfoilParams[
    numberInPassport].fileAirfoil);
79
80     PrintHeaderToTextFile(newForcesFile, "currentStep     currentTime     Fx     Fy
     Mz");
81
82     newForcesFileCsv << "t,Fx,Fy,Mz";
83
84     newForcesFile.close();
85     newForcesFile.clear();
86
87     newForcesFileCsv.close();
88     newForcesFileCsv.clear();
89
90 }//GenerateForcesHeader()
```

Граф вызовов:



Граф вызова функции:



10.9.3.5 GenerateForcesString()

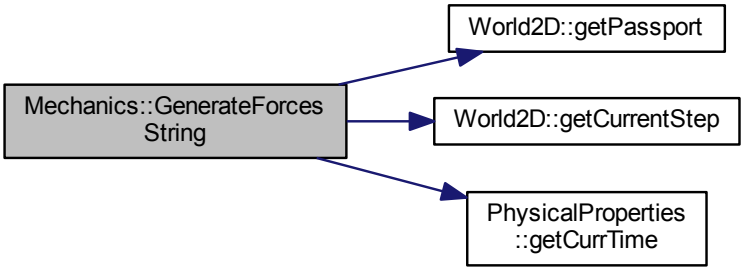
void Mechanics::GenerateForcesString ( )

Сохранение строки со статистикой в файл нагрузок

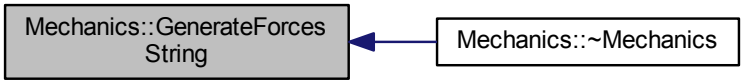
См. определение в файле Mechanics.cpp строка 121

```
{
    std::stringstream forceFileName, forceFileNameCsv;
    forceFileName << W.getPassport().dir << "forces-airfoil-" << numberInPassport;
    forceFileNameCsv << W.getPassport().dir << "forces-airfoil-" << numberInPassport << ".csv";

    std::ofstream forcesFile(forceFileName.str(), std::ios::app);
    forcesFile << std::endl << W.getCurrentStep() << "  " << W.getPassport().physicalProperties.getCurrTime() << " " << hydroDynamForce[0] << "  " << hydroDynamForce[1] << " " << hydroDynamMoment;
    forcesFile.close();

    std::ofstream forcesFileCsv(forceFileNameCsv.str(), std::ios::app);
    forcesFileCsv << std::endl << W.getPassport().physicalProperties.getCurrTime() << "," << hydroDynamForce[0] << "," << hydroDynamForce[1] << "," << hydroDynamMoment;
    forcesFileCsv.close();
}//GenerateForcesString()
```

Граф вызовов:



Граф вызова функции:



10.9.3.6 GeneratePositionHeader()

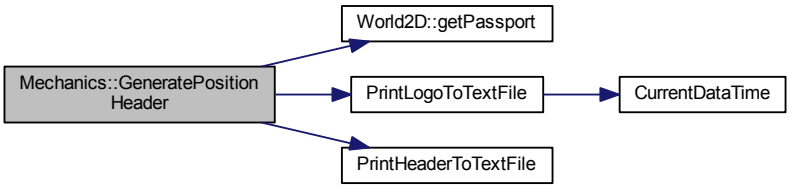
void Mechanics::GeneratePositionHeader ( )

Генерация заголовка файла положения профиля

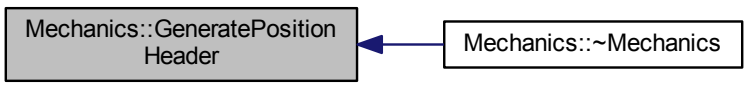
См. определение в файле Mechanics.cpp строка 94

```
95 {
96     if (isMoves)
97     {
98         std::stringstream positionFileName, positionFileNameCsv;
99         positionFileName << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport;
100         positionFileNameCsv << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport << ".csv";
101
102         std::ofstream newPositionFile(positionFileName.str());
103         std::ofstream newPositionFileCsv(positionFileNameCsv.str());
104
105         PrintLogoToTextFile(newPositionFile, positionFileName.str(), "Position of the
     airfoil " + W.getPassport().airfoilParams[numberInPassport].
    fileAirfoil);
106
107         PrintHeaderToTextFile(newPositionFile, "currentStep     currentTime     x
     y     phi     Vx     Vy     w");
108
109         newPositionFileCsv << "t,x,y,phi,Vx,Vy,w";
110
111         newPositionFile.close();
112         newPositionFile.clear();
113
114         newPositionFileCsv.close();
115         newPositionFileCsv.clear();
116     }
117 }//GeneratePositionHeader()
```

Граф вызовов:



Граф вызова функции:



10.9.3.7 GeneratePositionString()

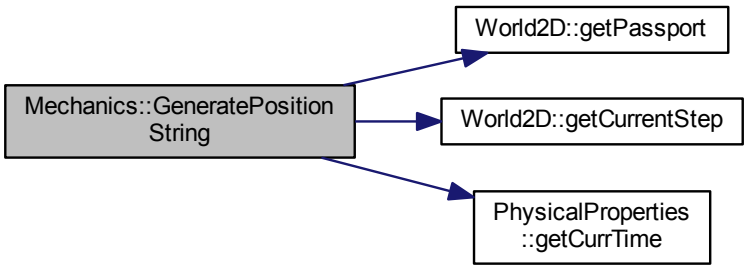
void Mechanics::GeneratePositionString ( )

Сохранение строки со статистикой в файл нагрузок

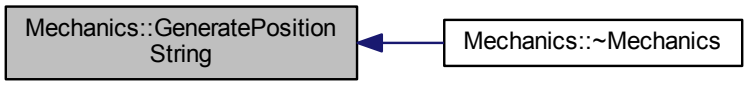
См. определение в файле Mechanics.cpp строка 138

```
{
    if (isMoves)
    {
        std::stringstream positionFileName, positionFileNameCsv;
        positionFileName << W.getPassport().dir << "position-airfoil-" << numberInPassport;
        positionFileNameCsv << W.getPassport().dir << "position-airfoil-" << numberInPassport << ".csv";

        std::ofstream forcesFile(positionFileName.str(), std::ios::app);
        forcesFile << std::endl << W.getCurrentStep() << "   " << W.getPassport().physicalProperties.getCurrTime() << "  " << afl.rcm[0] << "   " << afl.rcm[1] << "  " << Phi << "    " << Vcm[0] << " " << Vcm[1] << "  " << Wcm;
        forcesFile.close();

        std::ofstream forcesFileCsv(positionFileNameCsv.str(), std::ios::app);
        forcesFileCsv << std::endl << W.getPassport().physicalProperties.getCurrTime() << "," << afl.rcm[0] << "," << afl.rcm[1] << "," << Phi << "," << Vcm[0] << "," << Vcm[1] << "," << Wcm;
        forcesFileCsv.close();
    }
}
```

Граф вызовов:



Граф вызова функции:



10.9.3.8 GetHydroDynamForce()

virtual void Mechanics::GetHydroDynamForce (
timePeriod & time) [pure virtual]

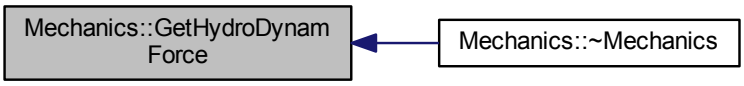
Вычисление гидродинамической силы, действующей на профиль

Аргументы

out	time	ссылка на промежуток времени — пару чисел (время начала и время конца операции)

Замещается в MechanicsRigidRotateMon, MechanicsRigidOscillPart, MechanicsRigidOscillMon, MechanicsRigidGivenLaw и MechanicsRigidImmovable.

Граф вызова функции:



10.9.3.9 Move()

virtual void Mechanics::Move ( ) [pure virtual]

Перемещение профиля в соответствии с законом

Замещается в MechanicsRigidRotateMon, MechanicsRigidOscillPart, MechanicsRigidOscillMon, MechanicsRigidGivenLaw и MechanicsRigidImmovable.

Граф вызова функции:

10.9.3.10 PositionOfAirfoilRcm()

virtual Point2D Mechanics::PositionOfAirfoilRcm (
double currTime) [pure virtual]

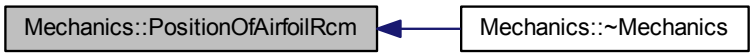
Вычисление положения центра масс профиля

Аргументы

in	currTime	текущее время

Замещается в MechanicsRigidRotateMon, MechanicsRigidOscillPart, MechanicsRigidOscillMon, MechanicsRigidGivenLaw и MechanicsRigidImmovable.

Граф вызова функции:



10.9.3.11 ReadParametersFromDictionary()

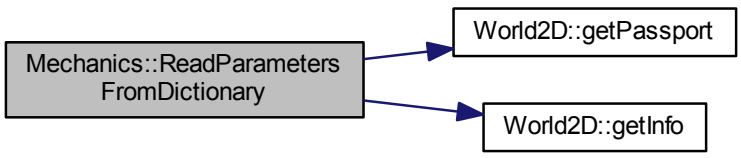
void Mechanics::ReadParametersFromDictionary ( ) [private]

Парсинг списка параметров механической системы

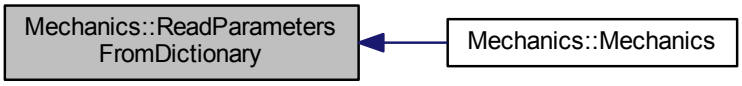
См. определение в файле Mechanics.cpp строка 60

```
61 {
62     std::stringstream ss(W.getPassport().airfoilParams[
    afl.numberInPassport].mechanicalSystemParameters);
63
64     mechParamsParser.reset(new StreamParser(W.
    getInfo(), "mechanical parser", ss));
65 }//ReadParametersFromDictionary()
```

Граф вызовов:



Граф вызова функции:



10.9.3.12 ReadSpecificParametersFromDictionary()

virtual void Mechanics::ReadSpecificParametersFromDictionary ( ) [pure virtual]

Чтение параметров конкретной механической системы

Замещается в MechanicsRigidRotateMon, MechanicsRigidOscillPart, MechanicsRigidOscillMon, MechanicsRigidGivenLaw и MechanicsRigidImmovable.

10.9.3.13 SolutionToMechanicalSystem()

virtual void Mechanics::SolutionToMechanicalSystem (
Eigen::VectorXd & col) [pure virtual]

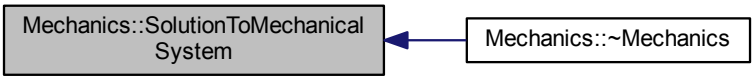
Извлечение из решения параметров механической системы

Аргументы

in	col	

Замещается в MechanicsRigidRotateMon, MechanicsRigidOscillPart, MechanicsRigidOscillMon, MechanicsRigidGivenLaw и MechanicsRigidImmovable.

Граф вызова функции:



10.9.3.14 VeloOfAirfoilPanels()

virtual void Mechanics::VeloOfAirfoilPanels (
double currTime) [pure virtual]

Вычисление скоростей начал панелей

Аргументы

in	currTime	текущее время

Замещается в MechanicsRigidRotateMon, MechanicsRigidOscillPart, MechanicsRigidOscillMon, MechanicsRigidGivenLaw и MechanicsRigidImmovable.

Граф вызова функции:

10.9.3.15 VeloOfAirfoilRcm()

virtual Point2D Mechanics::VeloOfAirfoilRcm (
double currTime) [pure virtual]

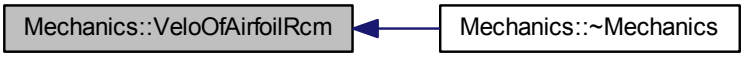
Вычисление скорости центра масс профиля

Аргументы

in	currTime	текущее время

Замещается в MechanicsRigidRotateMon, MechanicsRigidOscillPart, MechanicsRigidOscillMon, MechanicsRigidGivenLaw и MechanicsRigidImmovable.

Граф вызова функции:



10.9.4 Данные класса

10.9.4.1 afl

Airfoil& Mechanics::afl [protected]

Константная ссылка на профиль
инициализируется автоматом в конструкторе, при помощи const_cast

Предупреждения

использует const_cast для получения неконстантной ссылки

См. определение в файле Mechanics.h строка 83

10.9.4.2 boundary

const Boundary& Mechanics::boundary [protected]

Константная ссылка на граничное условие
инициализируется автоматом в конструкторе

См. определение в файле Mechanics.h строка 87

10.9.4.3 degOfFreedom

const size_t Mechanics::degOfFreedom

Количество степеней свободы

См. определение в файле Mechanics.h строка 125

10.9.4.4 hydroDynamForce

Point2D Mechanics::hydroDynamForce

Вектор гидродинамической силы и момент, действующие на профиль

См. определение в файле Mechanics.h строка 128

10.9.4.5 hydroDynamMoment

double Mechanics::hydroDynamMoment

См. определение в файле Mechanics.h строка 129

10.9.4.6 isDeform

const bool Mechanics::isDeform

Переменная, отвечающая за то, деформируется профиль или нет

См. определение в файле Mechanics.h строка 119

10.9.4.7 isMoves

const bool Mechanics::isMoves

Переменная, отвечающая за то, двигается профиль или нет

См. определение в файле Mechanics.h строка 116

10.9.4.8 isRotate

const bool Mechanics::isRotate

Переменная, отвечающая за то, может профиль вращаться или нет

См. определение в файле Mechanics.h строка 122

10.9.4.9 mechParamsParser

std::unique_ptr<StreamParser> Mechanics::mechParamsParser [protected]

Умный указатель на парсер параметров механической системы

См. определение в файле Mechanics.h строка 94

10.9.4.10 numberInPassport

const size_t Mechanics::numberInPassport [protected]

Номер профиля в паспорте

См. определение в файле Mechanics.h строка 78

10.9.4.11 Phi

double Mechanics::Phi [protected]

См. определение в файле Mechanics.h строка 106

10.9.4.12 Phi0

double Mechanics::Phi0 [protected]

См. определение в файле Mechanics.h строка 100

10.9.4.13 PhiOld

double Mechanics::PhiOld [protected]

См. определение в файле Mechanics.h строка 112

10.9.4.14 Rcm

Point2D Mechanics::Rcm [protected]

Текущие положение профиля

См. определение в файле Mechanics.h строка 106

10.9.4.15 Rcm0

const Point2D Mechanics::Rcm0 [protected]

Начальное положение профиля

См. определение в файле Mechanics.h строка 100

10.9.4.16 RcmOld

Point2D Mechanics::RcmOld [protected]

Текущие положение профиля

См. определение в файле Mechanics.h строка 112

10.9.4.17 Vcm

Point2D Mechanics::Vcm [protected]

Текущие скорость центра и угловая скорость

См. определение в файле Mechanics.h строка 103

10.9.4.18 Vcm0

const Point2D Mechanics::Vcm0 [protected]

Начальная скорость центра и угловая скорость

См. определение в файле Mechanics.h строка 97

10.9.4.19 VcmOld

Point2D Mechanics::VcmOld [protected]

Скорость и отклонение с предыдущего шага

См. определение в файле Mechanics.h строка 109

10.9.4.20 virtVortParams

const VortexesParams& Mechanics::virtVortParams [protected]

Константная ссылка на структуру с параметрами виртуального вихревого слоя для профиля инициализируется автоматом в конструкторе

См. определение в файле Mechanics.h строка 91

10.9.4.21 W

const World2D& Mechanics::W [protected]

Константная ссылка на решаемую задачу

См. определение в файле Mechanics.h строка 75

10.9.4.22 Wcm

double Mechanics::Wcm [protected]

См. определение в файле Mechanics.h строка 103

10.9.4.23 Wcm0

const double Mechanics::Wcm0 [protected]

См. определение в файле Mechanics.h строка 97

10.9.4.24 WcmOld

double Mechanics::WcmOld [protected]

См. определение в файле Mechanics.h строка 109

Объявления и описания членов классов находятся в файлах:

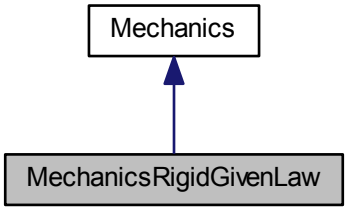
- Mechanics/Mechanics.h
- Mechanics/Mechanics.cpp

10.10 Класс MechanicsRigidGivenLaw

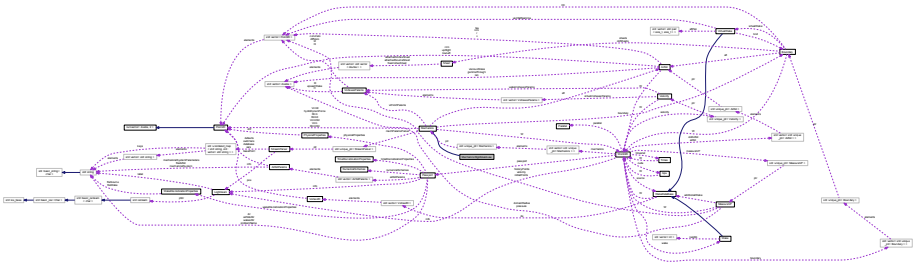
Класс, определяющий вид механической системы

#include <MechanicsRigidGivenLaw.h>

Граф наследования:MechanicsRigidGivenLaw:



Граф связей класса MechanicsRigidGivenLaw:



Открытые члены

- MechanicsRigidGivenLaw (const World2D &W_, size_t numberInPassport_)

  Конструктор
- ~MechanicsRigidGivenLaw ()

  Деструктор
- virtual void GetHydroDynamForce (timePeriod &time)

  Вычисление гидродинамической силы, действующей на профиль
- virtual Point2D VeloOfAirfoilRcm (double currTime)

  Вычисление скорости центра масс профиля
- virtual Point2D PositionOfAirfoilRcm (double currTime)

  Вычисление положения центра масс профиля
- virtual void VeloOfAirfoilPanels (double currTime)

  Вычисление скоростей начал панелей
- virtual void ReadSpecificParametersFromDictionary ()

  Чтение параметров конкретной механической системы
- virtual void FillMechanicsRowsAndCross (Eigen::MatrixXd &row, Eigen::MatrixXd &cross)

Заполнение строк в матрице, соответствующих механической системе. А также пересечений строк и столбцов, соответствующих мех. системе

- virtual void FillMechanicsRhs (std::vector< double > &rhs)

  Заполнение правых частей, соответствующих механической системе
- virtual void FillAtt (Eigen::MatrixXd &row, Eigen::MatrixXd &rhs)

  Заполнение правых частей (расщепленная схема) либо коэффициентов в матрице при скорости (монолитный подход), соответствующих присоединенным слоям
- virtual void SolutionToMechanicalSystem (Eigen::VectorXd &col)

  Извлечение из решения параметров механической системы
- virtual void Move ()

  Перемещение профиля в соответствии с законом
- void GenerateForcesHeader ()

  Генерация заголовка файла нагрузок
- void GeneratePositionHeader ()

  Генерация заголовка файла положения профиля
- void GenerateForcesString ()

  Сохранение строки со статистикой в файл нагрузок
- void GeneratePositionString ()

  Сохранение строки со статистикой в файл нагрузок

Открытые атрибуты

- const bool isMoves

  Переменная, отвечающая за то, двигается профиль или нет
- const bool isDeform

  Переменная, отвечающая за то, деформируется профиль или нет
- const bool isRotate

  Переменная, отвечающая за то, может профиль вращаться или нет
- const size_t degOfFreedom

  Количество степеней свободы
- Point2D hydroDynamForce

  Вектор гидродинамической силы и момент, действующие на профиль
- double hydroDynamMoment

Защищенные данные

- const World2D & W

  Константная ссылка на решаемую задачу
- const size_t numberInPassport

  Номер профиля в паспорте
- Airfoil & afl
- const Boundary & boundary
- const VortexesParams & virtVortParams
- std::unique_ptr< StreamParser > mechParamsParser

  Умный указатель на парсер параметров механической системы
- const Point2D Vcm0

  Начальная скорость центра и угловая скорость
- const double Wcm0
- const Point2D Rcm0

  Начальное положение профиля

- double Phi0
- Point2D Vcm

  Текущие скорость центра и угловая скорость
- double Wcm
- Point2D Rcm

  Текущие положение профиля
- double Phi
- Point2D VcmOld

  Скорость и отклонение с предыдущего шага
- double WcmOld
- Point2D RcmOld

  Текущие положение профиля
- double PhiOld

10.10.1 Подробное описание

Класс, определяющий вид механической системы

Жесткое тело, движущееся по заданному закону

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле MechanicsRigidGivenLaw.h строка 59

10.10.2 Конструктор(ы)

10.10.2.1 MechanicsRigidGivenLaw()

```
MechanicsRigidGivenLaw::MechanicsRigidGivenLaw (
            const World2D & W_,
            size_t numberInPassport_ )   [inline]
```

Конструктор

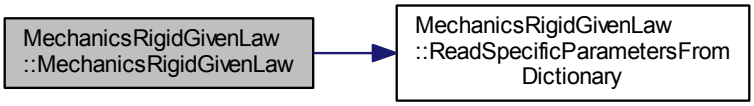
Аргументы

in	W_	константная ссылка на решаемую задачу
in	numberIn↩ Passport_	номер профиля в паспорте задачи

См. определение в файле MechanicsRigidGivenLaw.h строка 67

```
68        : Mechanics(W_, numberInPassport_, 0, true, false, false, { 0.0, 0.0 }, {0.0, 0.0}, 0.0, 0
    .0)
69    {
70        ReadSpecificParametersFromDictionary();
71    };
```

Граф вызовов:



10.10.2.2 ~MechanicsRigidGivenLaw()

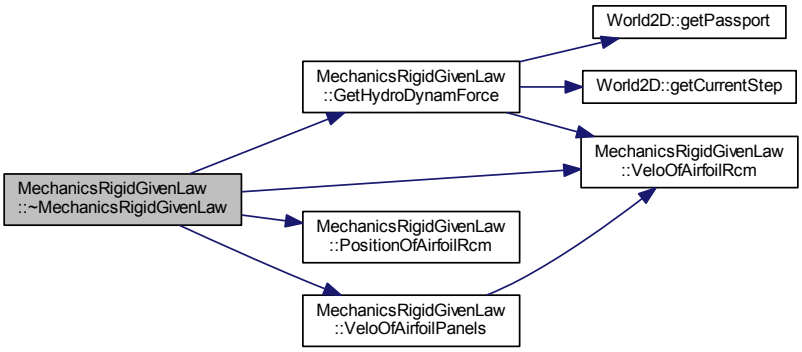
MechanicsRigidGivenLaw::~MechanicsRigidGivenLaw ( ) [inline]

Деструктор

См. определение в файле MechanicsRigidGivenLaw.h строка 75

```
75 { };
```

Граф вызовов:



10.10.3 Методы

10.10.3.1 FillAtt()

```
void MechanicsRigidGivenLaw::FillAtt (
            Eigen::MatrixXd & row,
            Eigen::MatrixXd & rhs )   [virtual]
```

Заполнение правых частей (расщепленная схема) либо коэффициентов в матрице при скорости (монолитный подход), соответствующих присоединенным слоям

Аргументы

in	row	строки (количество строк = количество степеней свободы, количество элементов в строках = количество неизвестных в локальной матрице)
in	rhs	вектор правой части для механики одного профиля (количество элементов = количеству степеней свободы)

Замещает Mechanics.

См. определение в файле MechanicsRigidGivenLaw.cpp строка 109

```
110 {
111     for (int i = 0; i < afl.np; ++i)
112         rhs(i, 0) = /*12*/0.5 * 0.5 * (afl.v[i] + afl.v[i+1])*afl.tau[i];
113 }
```

Граф вызова функции:

10.10.3.2 FillMechanicsRhs()

```
virtual void MechanicsRigidGivenLaw::FillMechanicsRhs (
            std::vector< double > & rhs )   [inline], [virtual]
```

Заполнение правых частей, соответствующих механической системе

Аргументы

in	rhs	вектор правой части для механики одного профиля (количество элементов = количеству степеней свободы)

Замещает Mechanics.

См. определение в файле MechanicsRigidGivenLaw.h строка 87

```
87 {};
```

Граф вызовов:

10.10.3.3 FillMechanicsRowsAndCross()

```
virtual void MechanicsRigidGivenLaw::FillMechanicsRowsAndCross (
            Eigen::MatrixXd & row,
            Eigen::MatrixXd & cross )   [inline], [virtual]
```

Заполнение строк в матрице, соответствующих механической системе. А также пересечений строк и столбцов, соответствующих мех. системе

Аргументы

in	row	строки (количество строк = количество степеней свободы, количество элементов в строках = количество неизвестных в локальной матрице)
in	cross	пересечения (количество столбцов = количество степеней свободы, количество элементов в столбцах = количество степеней свободы)

Замещает Mechanics.

См. определение в файле MechanicsRigidGivenLaw.h строка 85

```
85 {};
```

10.10.3.4 GenerateForcesHeader()

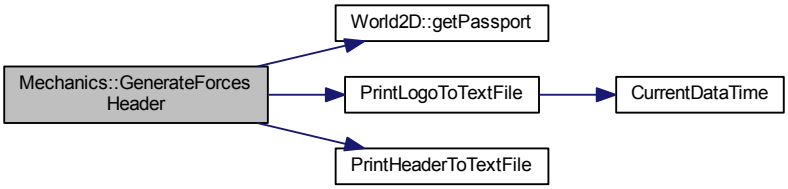
void Mechanics::GenerateForcesHeader ( ) [inherited]

Генерация заголовка файла нагрузок

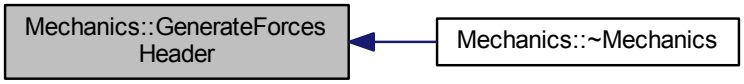
См. определение в файле Mechanics.cpp строка 69

```
70 {
71     std::stringstream forceFileName, forceFileNameCsv;
72     forceFileName << W.getPassport().dir << "forces-airfoil-" <<
    numberInPassport;
73     forceFileNameCsv << W.getPassport().dir << "forces-airfoil-" <<
    numberInPassport << ".csv";
74
75     std::ofstream newForcesFile(forceFileName.str());
76     std::ofstream newForcesFileCsv(forceFileNameCsv.str());
77
78     PrintLogoToTextFile(newForcesFile, forceFileName.str(), "Hydrodynamic loads for the
     airfoil " + W.getPassport().airfoilParams[
    numberInPassport].fileAirfoil);
79
80     PrintHeaderToTextFile(newForcesFile, "currentStep     currentTime     Fx     Fy
     Mz");
81
82     newForcesFileCsv << "t,Fx,Fy,Mz";
83
84     newForcesFile.close();
85     newForcesFile.clear();
86
87     newForcesFileCsv.close();
88     newForcesFileCsv.clear();
89
90 }//GenerateForcesHeader()
```

Граф вызовов:



Граф вызова функции:



10.10.3.5 GenerateForcesString()

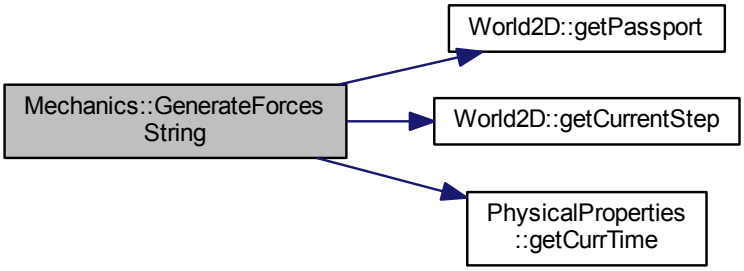
void Mechanics::GenerateForcesString ( ) [inherited]

Сохранение строки со статистикой в файл нагрузок

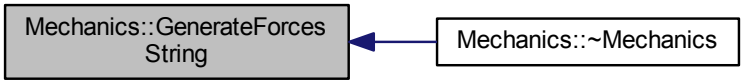
См. определение в файле Mechanics.cpp строка 121

```
{
    std::stringstream forceFileName, forceFileNameCsv;
    forceFileName << W.getPassport().dir << "forces-airfoil-" << numberInPassport;
    forceFileNameCsv << W.getPassport().dir << "forces-airfoil-" << numberInPassport << ".csv";

    std::ofstream forcesFile(forceFileName.str(), std::ios::app);
    forcesFile << std::endl << W.getCurrentStep() << "  " << W.getPassport().physicalProperties.getCurrTime() << " " << hydroDynamForce[0] << "  " << hydroDynamForce[1] << " " << hydroDynamMoment;
    forcesFile.close();

    std::ofstream forcesFileCsv(forceFileNameCsv.str(), std::ios::app);
    forcesFileCsv << std::endl << W.getPassport().physicalProperties.getCurrTime() << "," << hydroDynamForce[0] << "," << hydroDynamForce[1] << "," << hydroDynamMoment;
    forcesFileCsv.close();
}//GenerateForcesString()
```

Граф вызовов:



Граф вызова функции:



10.10.3.6 GeneratePositionHeader()

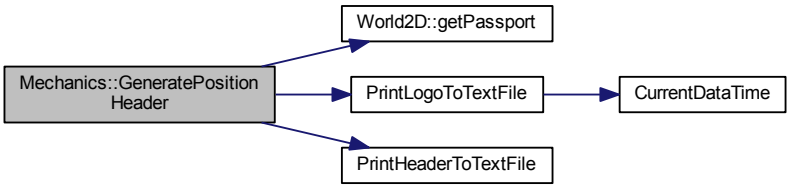
void Mechanics::GeneratePositionHeader ( ) [inherited]

Генерация заголовка файла положения профиля

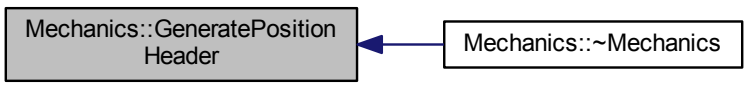
См. определение в файле Mechanics.cpp строка 94

```
95 {
96     if (isMoves)
97     {
98         std::stringstream positionFileName, positionFileNameCsv;
99         positionFileName << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport;
100         positionFileNameCsv << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport << ".csv";
101
102         std::ofstream newPositionFile(positionFileName.str());
103         std::ofstream newPositionFileCsv(positionFileNameCsv.str());
104
105         PrintLogoToTextFile(newPositionFile, positionFileName.str(), "Position of the
     airfoil " + W.getPassport().airfoilParams[numberInPassport].
    fileAirfoil);
106
107         PrintHeaderToTextFile(newPositionFile, "currentStep     currentTime     x
     y     phi     Vx     Vy     w");
108
109         newPositionFileCsv << "t,x,y,phi,Vx,Vy,w";
110
111         newPositionFile.close();
112         newPositionFile.clear();
113
114         newPositionFileCsv.close();
115         newPositionFileCsv.clear();
116     }
117 }//GeneratePositionHeader()
```

Граф вызовов:



Граф вызова функции:



10.10.3.7 GeneratePositionString()

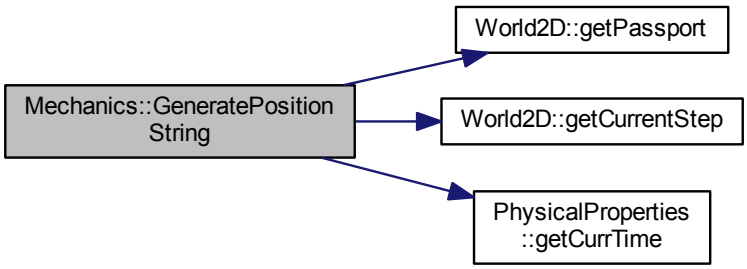
void Mechanics::GeneratePositionString ( ) [inherited]

Сохранение строки со статистикой в файл нагрузок

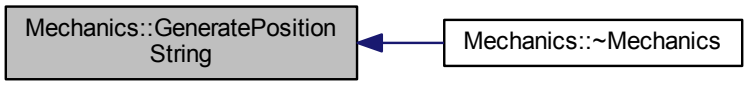
См. определение в файле Mechanics.cpp строка 138

```
139 {
140     if (isMoves)
141     {
142         std::stringstream positionFileName, positionFileNameCsv;
143         positionFileName << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport;
144         positionFileNameCsv << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport << ".csv";
145
146         std::ofstream forcesFile(positionFileName.str(), std::ios::app);
147         forcesFile << std::endl << W.getCurrentStep() << "   " <<
    W.getPassport().physicalProperties.getCurrTime() << "  " <<
    afl.rcm[0] << "   " << afl.rcm[1] << "  " << Phi << "    " << Vcm[0] << " " <<
    Vcm[1] << "  " << Wcm;
148         forcesFile.close();
149
150         std::ofstream forcesFileCsv(positionFileNameCsv.str(), std::ios::app);
151         forcesFileCsv << std::endl << W.getPassport().
    physicalProperties.getCurrTime() << "," << afl.
    rcm[0] << "," << afl.rcm[1] << "," << Phi << "," << Vcm[0] << "," <<
    Vcm[1] << "," << Wcm;
152         forcesFileCsv.close();
153     }
154 }
```

Граф вызовов:



Граф вызова функции:



10.10.3.8 GetHydroDynamForce()

void MechanicsRigidGivenLaw::GetHydroDynamForce (
timePeriod & time) [virtual]

Вычисление гидродинамической силы, действующей на профиль

Аргументы

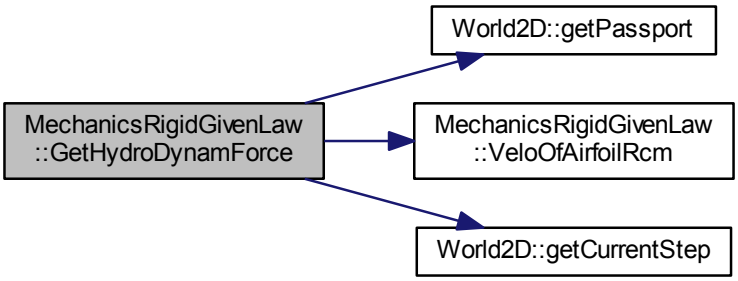
out	time	ссылка на промежуток времени — пару чисел (время начала и время конца операции)

Замещает Mechanics.

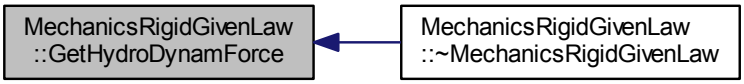
См. определение в файле MechanicsRigidGivenLaw.cpp строка 46

```
47 {
48     time.first = omp_get_wtime();
49
50     const double& dt = W.getPassport().timeDiscretizationProperties
    .dt;
51
52     hydroDynamForce = { 0.0, 0.0 };
53     hydroDynamMoment = 0.0;
54
55     Point2D hDFGam = { 0.0, 0.0 };   //гидродинамические силы, обусловленные Gamma_k
56     Point2D hDFdelta = { 0.0, 0.0 }; //гидродинамические силы, обусловленные delta_k
57     double hDMdelta = 0.0;
58
59     Point2D deltaVstep = VeloOfAirfoilRcm(W.
    getCurrentStep() * dt) - VeloOfAirfoilRcm((std::max(
    W.getCurrentStep(), (size_t)1) - 1) * dt);
60
61     for (size_t i = 0; i < afl.np; ++i)
62     {
63
64         double deltaK = boundary.sheets.freeVortexSheet[i][0] *
    afl.len[i] - afl.gammaThrough[i] + deltaVstep * afl.tau[i] *
    afl.len[i];
65         Point2D rK = 0.5 * (afl.r[i + 1] + afl.r[i]) - afl.rcm;
66
67         hDFdelta += deltaK * Point2D({ -rK[1], rK[0] });
68         hDMdelta += 0.5 * deltaK * (rK * rK);
69     }
70
71     hydroDynamForce = hDFdelta * (1.0 / dt);
72     hydroDynamMoment = hDMdelta / dt;
73
74
75     time.second = omp_get_wtime();
76 }// GetHydroDynamForce(...)
```

Граф вызовов:



Граф вызова функции:



10.10.3.9 Move()

void MechanicsRigidGivenLaw::Move ( ) [virtual]

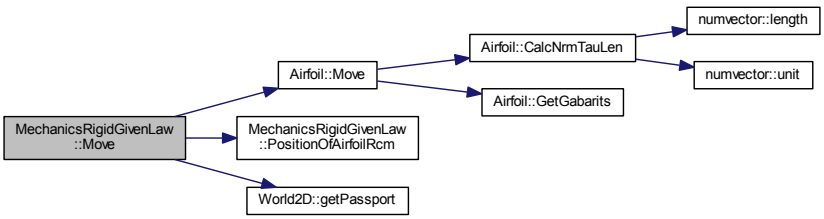
Перемещение профиля в соответствии с законом

Замещает Mechanics.

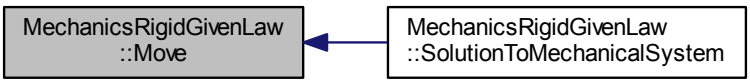
См. определение в файле MechanicsRigidGivenLaw.cpp строка 102

```
103 {
104     //Point2D airfoilVelo = VeloOfAirfoilRcm(W.getPassport().physicalProperties.currTime);
105     Point2D aflRcmOld = afl.rcm;
106     afl.Move(PositionOfAirfoilRcm(W.getPassport().
    physicalProperties.currTime) - aflRcmOld);
107 }
```

Граф вызовов:



Граф вызова функции:



10.10.3.10 PositionOfAirfoilRcm()

Point2D MechanicsRigidGivenLaw::PositionOfAirfoilRcm (
double currTime) [virtual]

Вычисление положения центра масс профиля

Аргументы

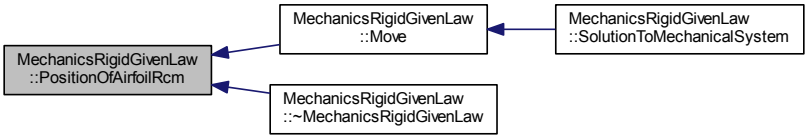
in	currTime	текущее время

Замещает Mechanics.

См. определение в файле MechanicsRigidGivenLaw.cpp строка 87

```
88 {
89     if (currTime < 10.0)
90         return { -currTime / 10.0 * currTime * 0.5, 0.0 };
91     else return { -5.0 - (currTime - 10.0), 0.0 };
92 }//PositionOfAirfoilRcm(...)
```

Граф вызова функции:



10.10.3.11 ReadSpecificParametersFromDictionary()

virtual void MechanicsRigidGivenLaw::ReadSpecificParametersFromDictionary ( ) [inline], [virtual]

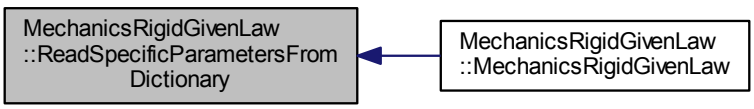
Чтение параметров конкретной механической системы

Замещает Mechanics.

См. определение в файле MechanicsRigidGivenLaw.h строка 82

```
82 {};
```

Граф вызова функции:



10.10.3.12 SolutionToMechanicalSystem()

virtual void MechanicsRigidGivenLaw::SolutionToMechanicalSystem (
Eigen::VectorXd & col) [inline], [virtual]

Извлечение из решения параметров механической системы

Аргументы

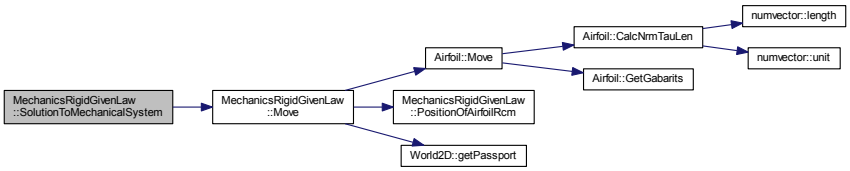
in	col	

Замещает Mechanics.

См. определение в файле MechanicsRigidGivenLaw.h строка 89

```
89 {};
```

Граф вызовов:



10.10.3.13 VeloOfAirfoilPanels()

void MechanicsRigidGivenLaw::VeloOfAirfoilPanels (
double currTime) [virtual]

Вычисление скоростей начал панелей

Аргументы

in	currTime	текущее время

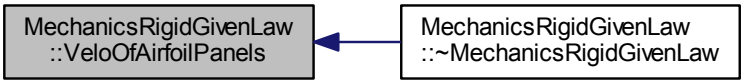
Замещает Mechanics.

См. определение в файле MechanicsRigidGivenLaw.cpp строка 95

```
96 {
97     Point2D veloRcm = VeloOfAirfoilRcm(currTime);
98     for (size_t i = 0; i < afl.v.size(); ++i)
99         afl.v[i] = veloRcm;
100 }//VeloOfAirfoilPanels(...)
```

Граф вызовов:

Граф вызова функции:



10.10.3.14 VeloOfAirfoilRcm()

Point2D MechanicsRigidGivenLaw::VeloOfAirfoilRcm (
double currTime) [virtual]

Вычисление скорости центра масс профиля

Аргументы

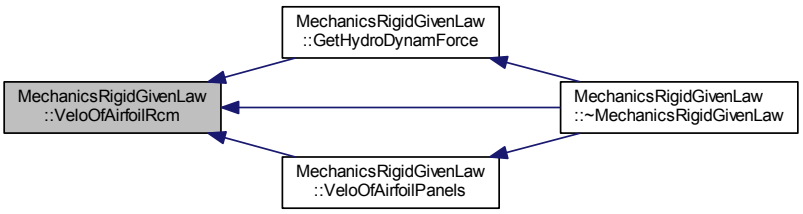
in	currTime	текущее время

Замещает Mechanics.

См. определение в файле MechanicsRigidGivenLaw.cpp строка 79

```
{
    if (currTime < 10.0)
        return { -currTime / 10.0, 0.0 };
    else return { -1.0, 0.0};
}//VeloOfAirfoilRcm(...)
```

Граф вызова функции:



10.10.4 Данные класса

10.10.4.1 afl

Airfoil& Mechanics::afl [protected], [inherited]

Константная ссылка на профиль
инициализируется автоматом в конструкторе, при помощи const_cast

Предупреждения

использует const_cast для получения неконстантной ссылки

См. определение в файле Mechanics.h строка 83

10.10.4.2 boundary

const Boundary& Mechanics::boundary [protected], [inherited]

Константная ссылка на граничное условие
инициализируется автоматом в конструкторе

См. определение в файле Mechanics.h строка 87

10.10.4.3 degOfFreedom

const size_t Mechanics::degOfFreedom [inherited]

Количество степеней свободы

См. определение в файле Mechanics.h строка 125

10.10.4.4 hydroDynamForce

Point2D Mechanics::hydroDynamForce [inherited]

Вектор гидродинамической силы и момент, действующие на профиль

См. определение в файле Mechanics.h строка 128

10.10.4.5 hydroDynamMoment

double Mechanics::hydroDynamMoment [inherited]

См. определение в файле Mechanics.h строка 129

10.10.4.6 isDeform

const bool Mechanics::isDeform [inherited]

Переменная, отвечающая за то, деформируется профиль или нет

См. определение в файле Mechanics.h строка 119

10.10.4.7 isMoves

const bool Mechanics::isMoves [inherited]

Переменная, отвечающая за то, двигается профиль или нет

См. определение в файле Mechanics.h строка 116

10.10.4.8 isRotate

const bool Mechanics::isRotate [inherited]

Переменная, отвечающая за то, может профиль вращаться или нет

См. определение в файле Mechanics.h строка 122

10.10.4.9 mechParamsParser

std::unique_ptr<StreamParser> Mechanics::mechParamsParser [protected], [inherited]

Умный указатель на парсер параметров механической системы

См. определение в файле Mechanics.h строка 94

10.10.4.10 numberInPassport

const size_t Mechanics::numberInPassport [protected], [inherited]

Номер профиля в паспорте

См. определение в файле Mechanics.h строка 78

10.10.4.11 Phi

double Mechanics::Phi [protected], [inherited]

См. определение в файле Mechanics.h строка 106

10.10.4.12 Phi0

double Mechanics::Phi0 [protected], [inherited]

См. определение в файле Mechanics.h строка 100

10.10.4.13 PhiOld

double Mechanics::PhiOld [protected], [inherited]

См. определение в файле Mechanics.h строка 112

10.10.4.14 Rcm

Point2D Mechanics::Rcm [protected], [inherited]

Текущие положение профиля

См. определение в файле Mechanics.h строка 106

10.10.4.15 Rcm0

const Point2D Mechanics::Rcm0 [protected], [inherited]

Начальное положение профиля

См. определение в файле Mechanics.h строка 100

10.10.4.16 RcmOld

Point2D Mechanics::RcmOld [protected], [inherited]

Текущие положение профиля

См. определение в файле Mechanics.h строка 112

10.10.4.17 Vcm

Point2D Mechanics::Vcm [protected], [inherited]

Текущие скорость центра и угловая скорость

См. определение в файле Mechanics.h строка 103

10.10.4.18 Vcm0

const Point2D Mechanics::Vcm0 [protected], [inherited]

Начальная скорость центра и угловая скорость

См. определение в файле Mechanics.h строка 97

10.10.4.19 VcmOld

Point2D Mechanics::VcmOld [protected], [inherited]

Скорость и отклонение с предыдущего шага

См. определение в файле Mechanics.h строка 109

10.10.4.20 virtVortParams

const VortexesParams& Mechanics::virtVortParams [protected], [inherited]

Константная ссылка на структуру с параметрами виртуального вихревого слоя для профиля инициализируется автоматом в конструкторе

См. определение в файле Mechanics.h строка 91

10.10.4.21 W

const World2D& Mechanics::W [protected], [inherited]

Константная ссылка на решаемую задачу

См. определение в файле Mechanics.h строка 75

10.10.4.22 Wcm

double Mechanics::Wcm [protected], [inherited]

См. определение в файле Mechanics.h строка 103

10.10.4.23 Wcm0

const double Mechanics::Wcm0 [protected], [inherited]

См. определение в файле Mechanics.h строка 97

10.10.4.24 WcmOld

double Mechanics::WcmOld [protected], [inherited]

См. определение в файле Mechanics.h строка 109

Объявления и описания членов классов находятся в файлах:

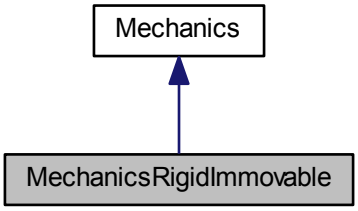
- Mechanics/MechanicsRigidGivenLaw.h
- Mechanics/MechanicsRigidGivenLaw.cpp

10.11 Класс MechanicsRigidImmovable

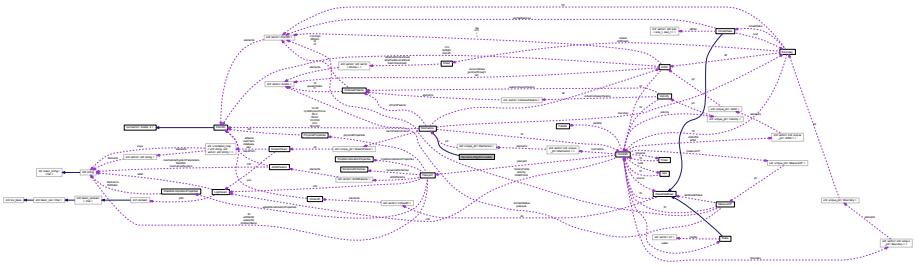
Класс, определяющий вид механической системы

#include <MechanicsRigidImmovable.h>

Граф наследования:MechanicsRigidImmovable:



Граф связей класса MechanicsRigidImmovable:



Открытые члены

- MechanicsRigidImmovable (const World2D &W_, size_t numberInPassport_)

  Конструктор
- ~MechanicsRigidImmovable ()

  Деструктор
- virtual void GetHydroDynamForce (timePeriod &time)

  Вычисление гидродинамической силы, действующей на профиль
- virtual Point2D VeloOfAirfoilRcm (double currTime)

  Вычисление скорости центра масс профиля
- virtual Point2D PositionOfAirfoilRcm (double currTime)

  Вычисление положения центра масс профиля
- virtual void VeloOfAirfoilPanels (double currTime)

  Вычисление скоростей начал панелей
- virtual void ReadSpecificParametersFromDictionary ()

  Чтение параметров конкретной механической системы
- virtual void FillMechanicsRowsAndCross (Eigen::MatrixXd &row, Eigen::MatrixXd &cross)

- virtual void FillMechanicsRhs (std::vector< double > &rhs)

  Заполнение правых частей, соответствующих механической системе
- virtual void FillAtt (Eigen::MatrixXd &row, Eigen::MatrixXd &rhs)

  Заполнение правых частей (расщепленная схема) либо коэффициентов в матрице при скорости (монолитный подход), соответствующих присоединенным слоям
- virtual void ComputeDisplacementRcm ()
- virtual void SolutionToMechanicalSystem (Eigen::VectorXd &col)

  Извлечение из решения параметров механической системы
- virtual void Move ()

  Перемещение профиля в соответствии с законом
- void GenerateForcesHeader ()

  Генерация заголовка файла нагрузок
- void GeneratePositionHeader ()

  Генерация заголовка файла положения профиля
- void GenerateForcesString ()

  Сохранение строки со статистикой в файл нагрузок
- void GeneratePositionString ()

  Сохранение строки со статистикой в файл нагрузок

Открытые атрибуты

- const bool isMoves

  Переменная, отвечающая за то, двигается профиль или нет
- const bool isDeform

  Переменная, отвечающая за то, деформируется профиль или нет
- const bool isRotate

  Переменная, отвечающая за то, может профиль вращаться или нет
- const size_t degOfFreedom

  Количество степеней свободы
- Point2D hydroDynamForce

  Вектор гидродинамической силы и момент, действующие на профиль
- double hydroDynamMoment

Защищенные данные

- const World2D & W

  Константная ссылка на решаемую задачу
- const size_t numberInPassport

  Номер профиля в паспорте
- Airfoil & afl
- const Boundary & boundary
- const VortexesParams & virtVortParams
- std::unique_ptr< StreamParser > mechParamsParser

  Умный указатель на парсер параметров механической системы
- const Point2D Vcm0

  Начальная скорость центра и угловая скорость
- const double Wcm0
- const Point2D Rcm0

  Начальное положение профиля
- double Phi0

- Point2D Vcm

  Текущие скорость центра и угловая скорость
- double Wcm
- Point2D Rcm

  Текущие положение профиля
- double Phi
- Point2D VcmOld

  Скорость и отклонение с предыдущего шага
- double WcmOld
- Point2D RcmOld

  Текущие положение профиля
- double PhiOld

10.11.1 Подробное описание

Класс, определяющий вид механической системы

Жесткое неподвижное тело

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле MechanicsRigidImmovable.h строка 59

10.11.2 Конструктор(ы)

10.11.2.1 MechanicsRigidImmovable()

```
MechanicsRigidImmovable::MechanicsRigidImmovable (
            const World2D & W_,
            size_t numberInPassport_ )   [inline]
```

Конструктор

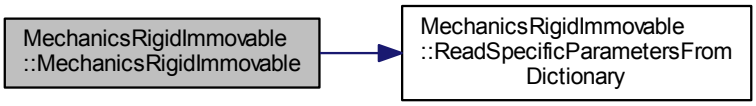
Аргументы

in	W_	константная ссылка на решаемую задачу
in	numberIn↩ Passport_	номер профиля в паспорте задачи

См. определение в файле MechanicsRigidImmovable.h строка 67

```
68        : Mechanics(W_, numberInPassport_, 0, false, false, false, { 0.0, 0.0 }, { 0.0, 0.0 }, 0.0
    , 0.0)
69    {
70        ReadSpecificParametersFromDictionary();
71    };
```

Граф вызовов:



10.11.2.2 ~MechanicsRigidImmovable()

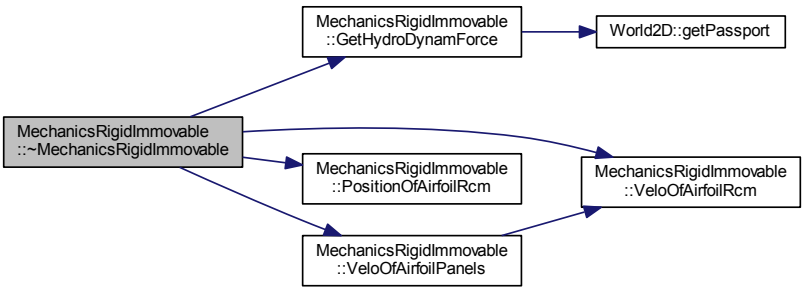
MechanicsRigidImmovable::~MechanicsRigidImmovable ( ) [inline]

Деструктор

См. определение в файле MechanicsRigidImmovable.h строка 74

```
74 {};
```

Граф вызовов:



10.11.3 Методы

10.11.3.1 ComputeDisplacementRcm()

virtual void MechanicsRigidImmovable::ComputeDisplacementRcm ( ) [inline], [virtual]

См. определение в файле MechanicsRigidImmovable.h строка 88

```
88 {};
```

10.11.3.2 FillAtt()

void MechanicsRigidImmovable::FillAtt (
Eigen::MatrixXd & row,
Eigen::MatrixXd & rhs) [virtual]

Заполнение правых частей (расщепленная схема) либо коэффициентов в матрице при скорости (монолитный подход), соответствующих присоединенным слоям

Аргументы

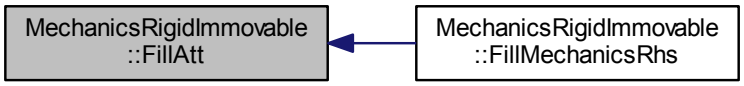
in	row	строки (количество строк = количество степеней свободы, количество элементов в строках = количество неизвестных в локальной матрице)
in	rhs	вектор правой части для механики одного профиля (количество элементов = количеству степеней свободы)

Замещает Mechanics.

См. определение в файле MechanicsRigidImmovable.cpp строка 93

```
94 {
95     for (int i = 0; i < afl.np; ++i)
96         rhs(i, 0) = /*12*/0.5 * 0.5 * (afl.v[i] + afl.v[i + 1])*afl.
    tau[i];
97 }
```

Граф вызова функции:



10.11.3.3 FillMechanicsRhs()

virtual void MechanicsRigidImmovable::FillMechanicsRhs (
std::vector< double > & rhs) [inline], [virtual]

Заполнение правых частей, соответствующих механической системе

Аргументы

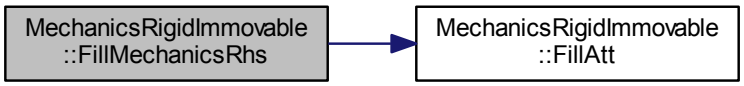
in	rhs	вектор правой части для механики одного профиля (количество элементов = количеству степеней свободы)

Замещает Mechanics.

См. определение в файле MechanicsRigidImmovable.h строка 86

```
86 {};
```

Граф вызовов:



10.11.3.4 FillMechanicsRowsAndCross()

virtual void MechanicsRigidImmovable::FillMechanicsRowsAndCross (
Eigen::MatrixXd & row,
Eigen::MatrixXd & cross) [inline], [virtual]

Необходимо сделать реализовать — не понял

Замещает Mechanics.

См. определение в файле MechanicsRigidImmovable.h строка 85

```
85 {};
```

10.11.3.5 GenerateForcesHeader()

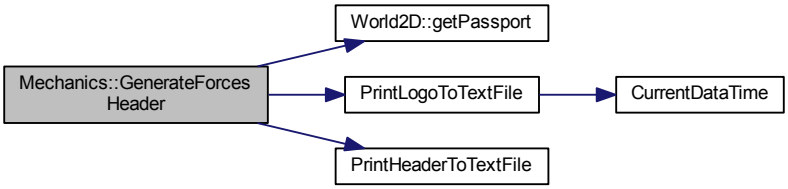
void Mechanics::GenerateForcesHeader ( ) [inherited]

Генерация заголовка файла нагрузок

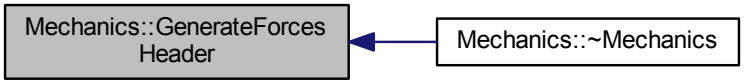
См. определение в файле Mechanics.cpp строка 69

```
70 {
71     std::stringstream forceFileName, forceFileNameCsv;
72     forceFileName << W.getPassport().dir << "forces-airfoil-" <<
    numberInPassport;
73     forceFileNameCsv << W.getPassport().dir << "forces-airfoil-" <<
    numberInPassport << ".csv";
74
75     std::ofstream newForcesFile(forceFileName.str());
76     std::ofstream newForcesFileCsv(forceFileNameCsv.str());
77
78     PrintLogoToTextFile(newForcesFile, forceFileName.str(), "Hydrodynamic loads for the
     airfoil " + W.getPassport().airfoilParams[
    numberInPassport].fileAirfoil);
79
80     PrintHeaderToTextFile(newForcesFile, "currentStep     currentTime     Fx     Fy
     Mz");
81
82     newForcesFileCsv << "t,Fx,Fy,Mz";
83
84     newForcesFile.close();
85     newForcesFile.clear();
86
87     newForcesFileCsv.close();
88     newForcesFileCsv.clear();
89
90 }//GenerateForcesHeader()
```

Граф вызовов:



Граф вызова функции:



10.11.3.6 GenerateForcesString()

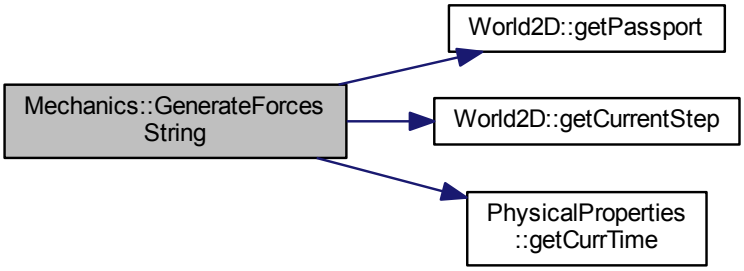
void Mechanics::GenerateForcesString ( ) [inherited]

Сохранение строки со статистикой в файл нагрузок

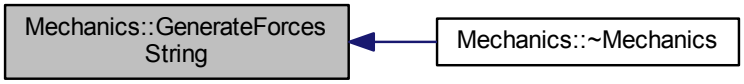
См. определение в файле Mechanics.cpp строка 121

```
{
    std::stringstream forceFileName, forceFileNameCsv;
    forceFileName << W.getPassport().dir << "forces-airfoil-" << numberInPassport;
    forceFileNameCsv << W.getPassport().dir << "forces-airfoil-" << numberInPassport << ".csv";

    std::ofstream forcesFile(forceFileName.str(), std::ios::app);
    forcesFile << std::endl << W.getCurrentStep() << "  " << W.getPassport().physicalProperties.getCurrTime() << " " << hydroDynamForce[0] << "  " << hydroDynamForce[1] << " " << hydroDynamMoment;
    forcesFile.close();

    std::ofstream forcesFileCsv(forceFileNameCsv.str(), std::ios::app);
    forcesFileCsv << std::endl << W.getPassport().physicalProperties.getCurrTime() << "," << hydroDynamForce[0] << "," << hydroDynamForce[1] << "," << hydroDynamMoment;
    forcesFileCsv.close();
}//GenerateForcesString()
```

Граф вызовов:



Граф вызова функции:



10.11.3.7 GeneratePositionHeader()

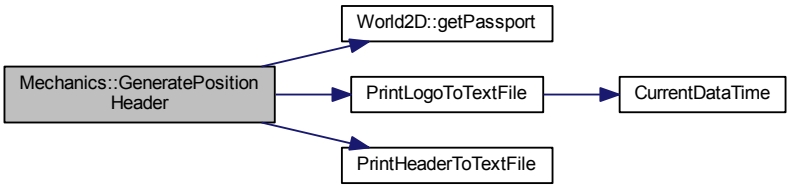
void Mechanics::GeneratePositionHeader ( ) [inherited]

Генерация заголовка файла положения профиля

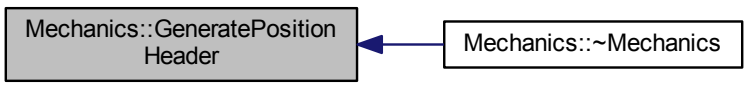
См. определение в файле Mechanics.cpp строка 94

```
95 {
96     if (isMoves)
97     {
98         std::stringstream positionFileName, positionFileNameCsv;
99         positionFileName << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport;
100         positionFileNameCsv << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport << ".csv";
101
102         std::ofstream newPositionFile(positionFileName.str());
103         std::ofstream newPositionFileCsv(positionFileNameCsv.str());
104
105         PrintLogoToTextFile(newPositionFile, positionFileName.str(), "Position of the
     airfoil " + W.getPassport().airfoilParams[numberInPassport].
    fileAirfoil);
106
107         PrintHeaderToTextFile(newPositionFile, "currentStep     currentTime     x
     y     phi     Vx     Vy     w");
108
109         newPositionFileCsv << "t,x,y,phi,Vx,Vy,w";
110
111         newPositionFile.close();
112         newPositionFile.clear();
113
114         newPositionFileCsv.close();
115         newPositionFileCsv.clear();
116     }
117 }//GeneratePositionHeader()
```

Граф вызовов:



Граф вызова функции:



10.11.3.8 GeneratePositionString()

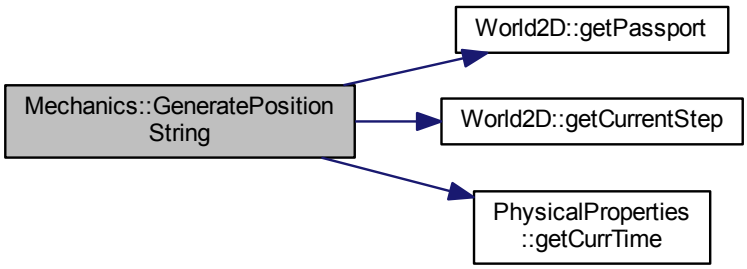
void Mechanics::GeneratePositionString ( ) [inherited]

Сохранение строки со статистикой в файл нагрузок

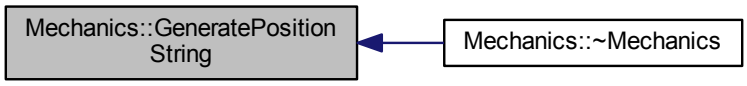
См. определение в файле Mechanics.cpp строка 138

```
{
    if (isMoves)
    {
        std::stringstream positionFileName, positionFileNameCsv;
        positionFileName << W.getPassport().dir << "position-airfoil-" << numberInPassport;
        positionFileNameCsv << W.getPassport().dir << "position-airfoil-" << numberInPassport << ".csv";

        std::ofstream forcesFile(positionFileName.str(), std::ios::app);
        forcesFile << std::endl << W.getCurrentStep() << "   " << W.getPassport().physicalProperties.getCurrTime() << "  " << afl.rcm[0] << "   " << afl.rcm[1] << "  " << Phi << "    " << Vcm[0] << " " << Vcm[1] << "  " << Wcm;
        forcesFile.close();

        std::ofstream forcesFileCsv(positionFileNameCsv.str(), std::ios::app);
        forcesFileCsv << std::endl << W.getPassport().physicalProperties.getCurrTime() << "," << afl.rcm[0] << "," << afl.rcm[1] << "," << Phi << "," << Vcm[0] << "," << Vcm[1] << "," << Wcm;
        forcesFileCsv.close();
    }
}
```

Граф вызовов:



Граф вызова функции:



10.11.3.9 GetHydroDynamForce()

```
void MechanicsRigidImmovable::GetHydroDynamForce (
            timePeriod & time )  [virtual]
```

Вычисление гидродинамической силы, действующей на профиль

Аргументы

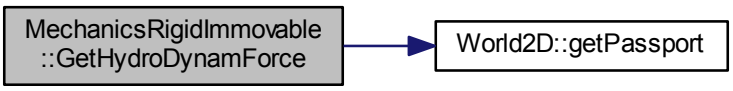
out	time	ссылка на промежуток времени — пару чисел (время начала и время конца операции)

Замещает Mechanics.

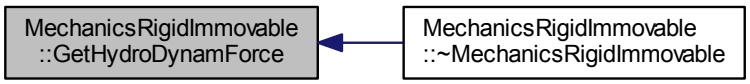
См. определение в файле MechanicsRigidImmovable.cpp строка 64

```
65 {
66     time.first = omp_get_wtime();
67
68     const double& dt = W.getPassport().timeDiscretizationProperties
    .dt;
69
70     hydroDynamForce = { 0.0, 0.0 };
71     hydroDynamMoment = 0.0;
72
73     Point2D hDFGam = { 0.0, 0.0 };   //гидродинамические силы, обусловленные Gamma_k
74     Point2D hDFdelta = { 0.0, 0.0 }; //гидродинамические силы, обусловленные delta_k
75     double hDMdelta = 0.0;
76
77     for (size_t i = 0; i < afl.np; ++i)
78     {
79         double deltaK =  boundary.sheets.freeVortexSheet[i][0] *
    afl.len[i] - afl.gammaThrough[i];
80         Point2D rK = 0.5 * (afl.r[i + 1] + afl.r[i]) - afl.rcm;
81
82         hDFdelta += deltaK * Point2D({ -rK[1], rK[0] });
83         hDMdelta += 0.5 * deltaK * (rK * rK);
84     }
85
86     hydroDynamForce =  hDFdelta * (1.0 / dt);
87     hydroDynamMoment = hDMdelta / dt;
88
89
90     time.second = omp_get_wtime();
91 }//GetHydroDynamForce(...)
```

Граф вызовов:



Граф вызова функции:



10.11.3.10 Move()

virtual void MechanicsRigidImmovable::Move ( ) [inline], [virtual]

Перемещение профиля в соответствии с законом

Замещает Mechanics.

См. определение в файле MechanicsRigidImmovable.h строка 90

```
90 {};
```

10.11.3.11 PositionOfAirfoilRcm()

Point2D MechanicsRigidImmovable::PositionOfAirfoilRcm (
double currTime) [virtual]

Вычисление положения центра масс профиля

Аргументы

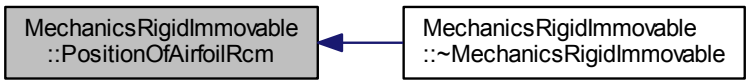
in	currTime	текущее время

Замещает Mechanics.

См. определение в файле MechanicsRigidImmovable.cpp строка 50

```
51 {
52     return afl.rcm;
53 }//PositionOfAirfoilRcm(...)
```

Граф вызова функции:



10.11.3.12 ReadSpecificParametersFromDictionary()

virtual void MechanicsRigidImmovable::ReadSpecificParametersFromDictionary ( ) [inline], [virtual]

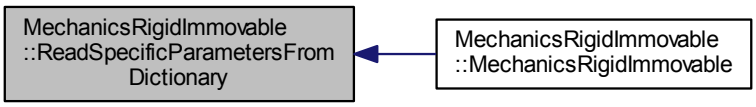
Чтение параметров конкретной механической системы

Замещает Mechanics.

См. определение в файле MechanicsRigidImmovable.h строка 82

```
82 {};
```

Граф вызова функции:



10.11.3.13 SolutionToMechanicalSystem()

virtual void MechanicsRigidImmovable::SolutionToMechanicalSystem (
Eigen::VectorXd & col) [inline], [virtual]

Извлечение из решения параметров механической системы

Аргументы

in	col	

Замещает Mechanics.

См. определение в файле MechanicsRigidImmovable.h строка 89

```
89 {};
```

10.11.3.14 VeloOfAirfoilPanels()

void MechanicsRigidImmovable::VeloOfAirfoilPanels (
double currTime) [virtual]

Вычисление скоростей начал панелей

Аргументы

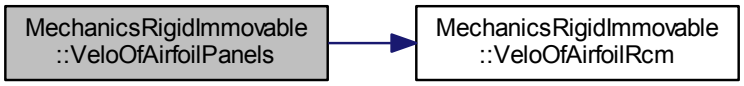
in	currTime	текущее время

Замещает Mechanics.

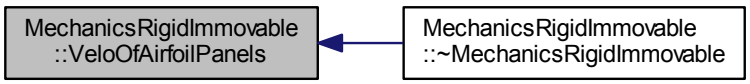
См. определение в файле MechanicsRigidImmovable.cpp строка 56

```
57 {
58     Point2D veloRcm = VeloOfAirfoilRcm(currTime);
59     for (size_t i = 0; i < afl.v.size(); ++i)
60         afl.v[i] = veloRcm;
61 }//VeloOfAirfoilPanels(...)
```

Граф вызовов:



Граф вызова функции:



10.11.3.15 VeloOfAirfoilRcm()

Point2D MechanicsRigidImmovable::VeloOfAirfoilRcm (
double currTime) [virtual]

Вычисление скорости центра масс профиля

Аргументы

in	currTime	текущее время

Замещает Mechanics.

См. определение в файле MechanicsRigidImmovable.cpp строка 44

```
45 {
46     return{ 0.0, 0.0 };
47 }//VeloOfAirfoilRcm(...)
```

Граф вызова функции:

10.11.4 Данные класса

10.11.4.1 afl

Airfoil& Mechanics::afl [protected], [inherited]

Константная ссылка на профиль
инициализируется автоматом в конструкторе, при помощи const_cast

Предупреждения

использует const_cast для получения неконстантной ссылки

См. определение в файле Mechanics.h строка 83

10.11.4.2 boundary

const Boundary& Mechanics::boundary [protected], [inherited]

Константная ссылка на граничное условие
инициализируется автоматом в конструкторе

См. определение в файле Mechanics.h строка 87

10.11.4.3 degOfFreedom

const size_t Mechanics::degOfFreedom [inherited]

Количество степеней свободы

См. определение в файле Mechanics.h строка 125

10.11.4.4 hydroDynamForce

Point2D Mechanics::hydroDynamForce [inherited]

Вектор гидродинамической силы и момент, действующие на профиль

См. определение в файле Mechanics.h строка 128

10.11.4.5 hydroDynamMoment

double Mechanics::hydroDynamMoment [inherited]

См. определение в файле Mechanics.h строка 129

10.11.4.6 isDeform

const bool Mechanics::isDeform [inherited]

Переменная, отвечающая за то, деформируется профиль или нет

См. определение в файле Mechanics.h строка 119

10.11.4.7 isMoves

const bool Mechanics::isMoves [inherited]

Переменная, отвечающая за то, двигается профиль или нет

См. определение в файле Mechanics.h строка 116

10.11.4.8 isRotate

const bool Mechanics::isRotate [inherited]

Переменная, отвечающая за то, может профиль вращаться или нет

См. определение в файле Mechanics.h строка 122

10.11.4.9 mechParamsParser

std::unique_ptr<StreamParser> Mechanics::mechParamsParser [protected], [inherited]

Умный указатель на парсер параметров механической системы

См. определение в файле Mechanics.h строка 94

10.11.4.10 numberInPassport

const size_t Mechanics::numberInPassport [protected], [inherited]

Номер профиля в паспорте

См. определение в файле Mechanics.h строка 78

10.11.4.11 Phi

double Mechanics::Phi [protected], [inherited]

См. определение в файле Mechanics.h строка 106

10.11.4.12 Phi0

double Mechanics::Phi0 [protected], [inherited]

См. определение в файле Mechanics.h строка 100

10.11.4.13 PhiOld

double Mechanics::PhiOld [protected], [inherited]

См. определение в файле Mechanics.h строка 112

10.11.4.14 Rcm

Point2D Mechanics::Rcm [protected], [inherited]

Текущие положение профиля

См. определение в файле Mechanics.h строка 106

10.11.4.15 Rcm0

const Point2D Mechanics::Rcm0 [protected], [inherited]

Начальное положение профиля

См. определение в файле Mechanics.h строка 100

10.11.4.16 RcmOld

Point2D Mechanics::RcmOld [protected], [inherited]

Текущие положение профиля

См. определение в файле Mechanics.h строка 112

10.11.4.17 Vcm

Point2D Mechanics::Vcm [protected], [inherited]

Текущие скорость центра и угловая скорость

См. определение в файле Mechanics.h строка 103

10.11.4.18 Vcm0

const Point2D Mechanics::Vcm0 [protected], [inherited]

Начальная скорость центра и угловая скорость

См. определение в файле Mechanics.h строка 97

10.11.4.19 VcmOld

Point2D Mechanics::VcmOld [protected], [inherited]

Скорость и отклонение с предыдущего шага

См. определение в файле Mechanics.h строка 109

10.11.4.20 virtVortParams

const VortexesParams& Mechanics::virtVortParams [protected], [inherited]

Константная ссылка на структуру с параметрами виртуального вихревого слоя для профиля инициализируется автоматом в конструкторе

См. определение в файле Mechanics.h строка 91

10.11.4.21 W

const World2D& Mechanics::W [protected], [inherited]

Константная ссылка на решаемую задачу

См. определение в файле Mechanics.h строка 75

10.11.4.22 Wcm

double Mechanics::Wcm [protected], [inherited]

См. определение в файле Mechanics.h строка 103

10.11.4.23 Wcm0

const double Mechanics::Wcm0 [protected], [inherited]

См. определение в файле Mechanics.h строка 97

10.11.4.24 WcmOld

double Mechanics::WcmOld [protected], [inherited]

См. определение в файле Mechanics.h строка 109

Объявления и описания членов классов находятся в файлах:

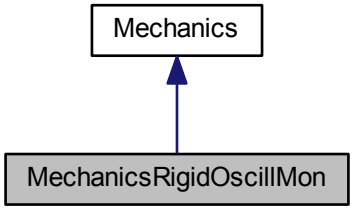
- Mechanics/MechanicsRigidImmovable.h
- Mechanics/MechanicsRigidImmovable.cpp

10.12 Класс MechanicsRigidOscillMon

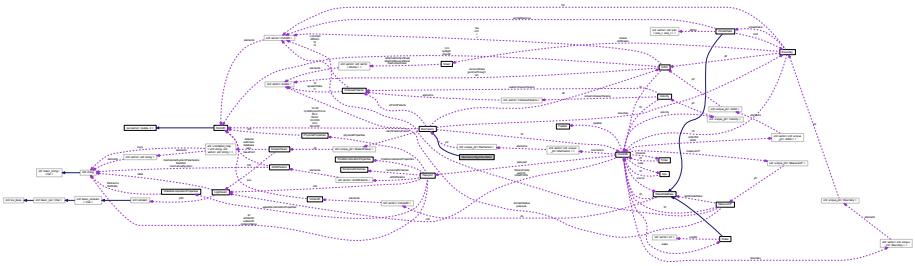
Класс, определяющий вид механической системы

#include <MechanicsRigidOscillMon.h>

Граф наследования:MechanicsRigidOscillMon:



Граф связей класса MechanicsRigidOscillMon:



Открытые члены

- MechanicsRigidOscillMon (const World2D &W_, size_t numberInPassport_)

  Конструктор
- ~MechanicsRigidOscillMon ()

  Деструктор
- virtual void GetHydroDynamForce (timePeriod &time)

  Вычисление гидродинамической силы, действующей на профиль
- virtual Point2D VeloOfAirfoilRcm (double currTime)

  Вычисление скорости центра масс профиля
- virtual Point2D PositionOfAirfoilRcm (double currTime)

  Вычисление положения центра масс профиля
- virtual void VeloOfAirfoilPanels (double currTime)

  Вычисление скоростей начал панелей
- virtual void ReadSpecificParametersFromDictionary ()

  Чтение параметров конкретной механической системы
- virtual void FillMechanicsRowsAndCross (Eigen::MatrixXd &row, Eigen::MatrixXd &cross)
- virtual void FillMechanicsRhs (std::vector< double > &rhs)

  Заполнение правых частей, соответствующих механической системе
- virtual void FillAtt (Eigen::MatrixXd &row, Eigen::MatrixXd &rhs)

  Заполнение правых частей (расщепленная схема) либо коэффициентов в матрице при скорости (монолитный подход), соответствующих присоединенным слоям
- virtual void SolutionToMechanicalSystem (Eigen::VectorXd &col)

  Извлечение из решения параметров механической системы
- virtual void Move ()

  Перемещение профиля в соответствии с законом
- void GenerateForcesHeader ()

  Генерация заголовка файла нагрузок
- void GeneratePositionHeader ()

  Генерация заголовка файла положения профиля
- void GenerateForcesString ()

  Сохранение строки со статистикой в файл нагрузок
- void GeneratePositionString ()

  Сохранение строки со статистикой в файл нагрузок

Открытые атрибуты

- const bool isMoves

  Переменная, отвечающая за то, двигается профиль или нет
- const bool isDeform

  Переменная, отвечающая за то, деформируется профиль или нет
- const bool isRotate

  Переменная, отвечающая за то, может профиль вращаться или нет
- const size_t degOfFreedom

  Количество степеней свободы
- Point2D hydroDynamForce

  Вектор гидродинамической силы и момент, действующие на профиль
- double hydroDynamMoment

Защищенные данные

- const World2D & W

  Константная ссылка на решаемую задачу
- const size_t numberInPassport

  Номер профиля в паспорте
- Airfoil & afl
- const Boundary & boundary
- const VortexesParams & virtVortParams
- std::unique_ptr< StreamParser > mechParamsParser

  Умный указатель на парсер параметров механической системы
- const Point2D Vcm0

  Начальная скорость центра и угловая скорость
- const double Wcm0
- const Point2D Rcm0

  Начальное положение профиля
- double Phi0
- Point2D Vcm

  Текущие скорость центра и угловая скорость
- double Wcm
- Point2D Rcm

  Текущие положение профиля
- double Phi
- Point2D VcmOld

  Скорость и отклонение с предыдущего шага
- double WcmOld
- Point2D RcmOld

  Текущие положение профиля
- double PhiOld

Закрытые данные

- double m
- const double b
- double k

10.12.1 Подробное описание

Класс, определяющий вид механической системы

Жесткое тело, движущееся по заданному закону

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле MechanicsRigidOscillMon.h строка 59

10.12.2 Конструктор(ы)

10.12.2.1 MechanicsRigidOscillMon()

```
MechanicsRigidOscillMon::MechanicsRigidOscillMon (
            const World2D & W_,
            size_t numberInPassport_ )   [inline]
```

Конструктор

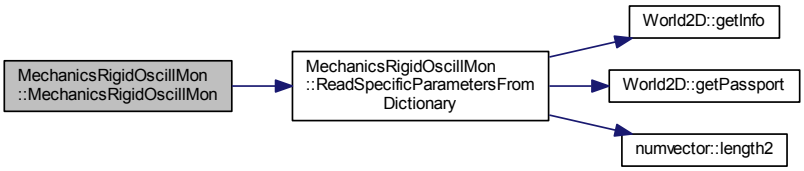
Аргументы

in	W_	константная ссылка на решаемую задачу
in	numberIn↩ Passport_	номер профиля в паспорте задачи

См. определение в файле MechanicsRigidOscillMon.h строка 78

```
79      : Mechanics(W_, numberInPassport_, 1, true, false, false, { 0.0, 0.0 }, { 0.0, 0.0 }, 0.0,
    0.0), m(39.15), b(0.0 * 0.731)
80    {
81
82      ReadSpecificParametersFromDictionary();
83    };
```

Граф вызовов:



10.12.2.2 ~MechanicsRigidOscillMon()

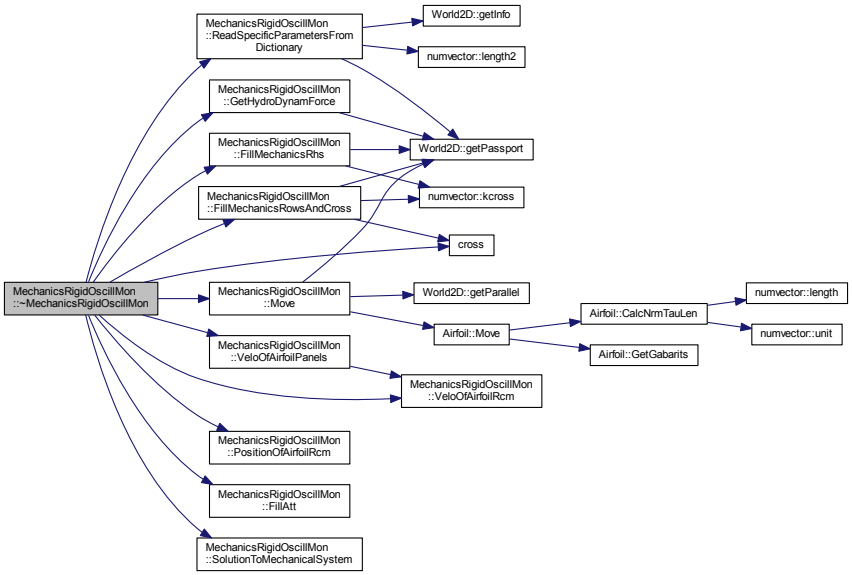
```
MechanicsRigidOscillMon::~MechanicsRigidOscillMon ( )   [inline]
```

Деструктор

См. определение в файле MechanicsRigidOscillMon.h строка 86

```
86 {};
```

Граф вызовов:



10.12.3 Методы

10.12.3.1 FillAtt()

void MechanicsRigidOscillMon::FillAtt (
Eigen::MatrixXd & row,
Eigen::MatrixXd & rhs) [virtual]

Заполнение правых частей (расщепленная схема) либо коэффициентов в матрице при скорости (монолитный подход), соответствующих присоединенным слоям

Аргументы

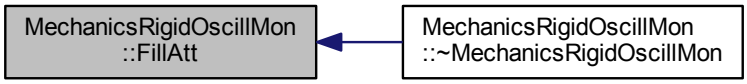
in	row	строки (количество строк = количество степеней свободы, количество элементов в строках = количество неизвестных в локальной матрице)
in	rhs	вектор правой части для механики одного профиля (количество элементов = количеству степеней свободы)

Замещает Mechanics.

См. определение в файле MechanicsRigidOscillMon.cpp строка 152

```
153 {
154     for (size_t i = 0; i < afl.np; ++i)
155         col(i, 0) = /*12*/ -afl.tau[i][1];
156 }
```

Граф вызова функции:



10.12.3.2 FillMechanicsRhs()

void MechanicsRigidOscillMon::FillMechanicsRhs (
std::vector< double > & rhs) [virtual]

Заполнение правых частей, соответствующих механической системе

Аргументы

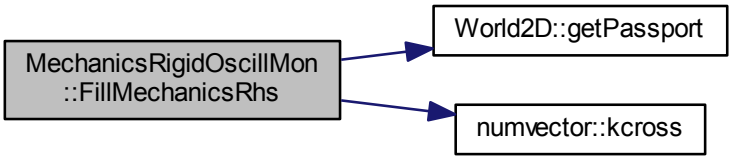
in	rhs	вектор правой части для механики одного профиля (количество элементов = количеству степеней свободы)

Замещает Mechanics.

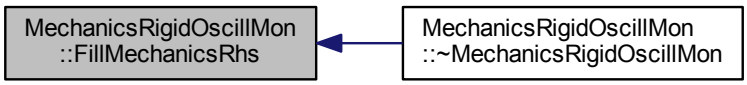
См. определение в файле MechanicsRigidOscillMon.cpp строка 141

```
142 {
143     rhs[0] = /*7*/-m * Vcm[1] + /*8*/ W.getPassport().
    timeDiscretizationProperties.dt * k * Rcm[1];
144
145     for (size_t loc_i = 0; loc_i < afl.np; ++loc_i)
146     {
147         Point2D riC = 0.5 * (afl.r[loc_i + 1] + afl.r[loc_i]) -
    afl.rcm;
148         rhs[0] += riC.kcross()[1] * (/*5*/afl.gammaThrough[loc_i] + /*6*/
    Vcm[1] * afl.tau[loc_i][1] * afl.len[loc_i]);
149     }
150 }
```

Граф вызовов:



Граф вызова функции:



10.12.3.3 FillMechanicsRowsAndCross()

```
void MechanicsRigidOscillMon::FillMechanicsRowsAndCross (
            Eigen::MatrixXd & row,
            Eigen::MatrixXd & cross )   [virtual]
```

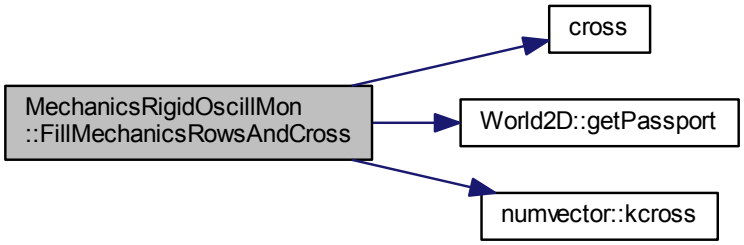
Необходимо сделать реализовать — не понял

Замещает Mechanics.

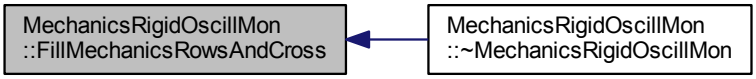
См. определение в файле MechanicsRigidOscillMon.cpp строка 126

```
{
    cross(0, 0) = -(/*3*/ m + /*4*/ W.getPassport().timeDiscretizationProperties.dt * b);

    Point2D riC;

    for (size_t loc_i = 0; loc_i < afl.np; ++loc_i)
    {
        riC = 0.5 * (afl.r[loc_i + 1] + afl.r[loc_i]) - afl.rcm;
        row(0, loc_i) = /*1*/ riC.kcross()[1] * afl.len[loc_i];
        cross(0, 0) += /*2*/ riC.kcross()[1] * afl.tau[loc_i][1] * afl.len[loc_i];
    }
}
```

Граф вызовов:



Граф вызова функции:



10.12.3.4 GenerateForcesHeader()

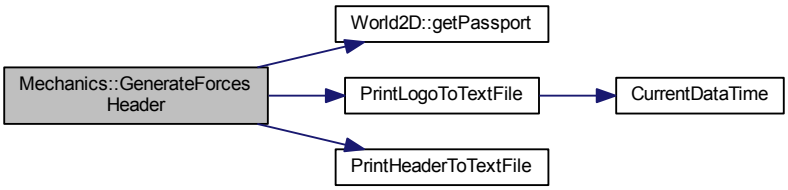
void Mechanics::GenerateForcesHeader ( ) [inherited]

Генерация заголовка файла нагрузок

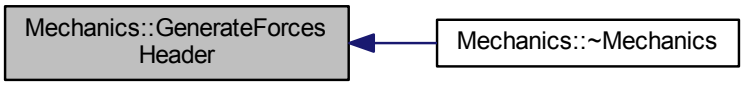
См. определение в файле Mechanics.cpp строка 69

```
{
  std::stringstream forceFileName, forceFileNameCsv;
  forceFileName << W.getPassport().dir << "forces-airfoil-" << numberInPassport;
  forceFileNameCsv << W.getPassport().dir << "forces-airfoil-" << numberInPassport << ".csv";

  std::ofstream newForcesFile(forceFileName.str());
  std::ofstream newForcesFileCsv(forceFileNameCsv.str());

  PrintLogoToTextFile(newForcesFile, forceFileName.str(), "Hydrodynamic loads for the airfoil " + W.getPassport().airfoilParams[numberInPassport].fileAirfoil);

  PrintHeaderToTextFile(newForcesFile, "currentStep    currentTime    Fx    Fy    Mz");

  newForcesFileCsv << "t,Fx,Fy,Mz";

  newForcesFile.close();
  newForcesFile.clear();

  newForcesFileCsv.close();
  newForcesFileCsv.clear();

}//GenerateForcesHeader()
```

Граф вызовов:



Граф вызова функции:



10.12.3.5 GenerateForcesString()

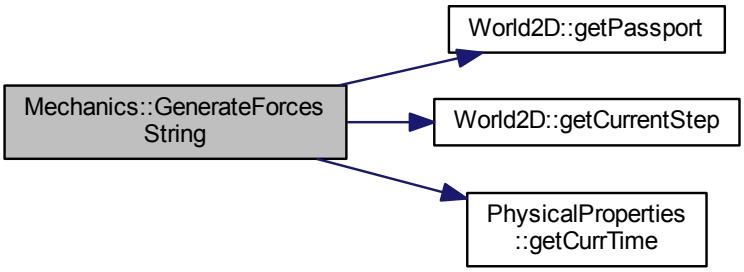
void Mechanics::GenerateForcesString ( ) [inherited]

Сохранение строки со статистикой в файл нагрузок

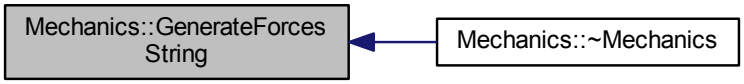
См. определение в файле Mechanics.cpp строка 121

```
{
    std::stringstream forceFileName, forceFileNameCsv;
    forceFileName << W.getPassport().dir << "forces-airfoil-" << numberInPassport;
    forceFileNameCsv << W.getPassport().dir << "forces-airfoil-" << numberInPassport << ".csv";

    std::ofstream forcesFile(forceFileName.str(), std::ios::app);
    forcesFile << std::endl << W.getCurrentStep() << "  " << W.getPassport().physicalProperties.getCurrTime() << " " << hydroDynamForce[0] << "  " << hydroDynamForce[1] << " " << hydroDynamMoment;
    forcesFile.close();

    std::ofstream forcesFileCsv(forceFileNameCsv.str(), std::ios::app);
    forcesFileCsv << std::endl << W.getPassport().physicalProperties.getCurrTime() << "," << hydroDynamForce[0] << "," << hydroDynamForce[1] << "," << hydroDynamMoment;
    forcesFileCsv.close();
}//GenerateForcesString()
```

Граф вызовов:



Граф вызова функции:



10.12.3.6 GeneratePositionHeader()

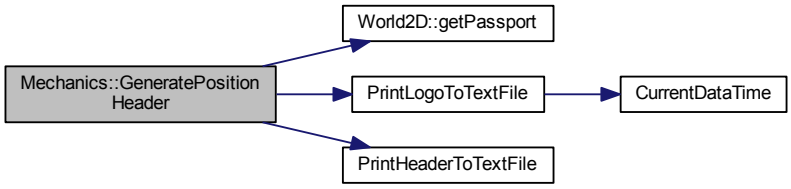
void Mechanics::GeneratePositionHeader ( ) [inherited]

Генерация заголовка файла положения профиля

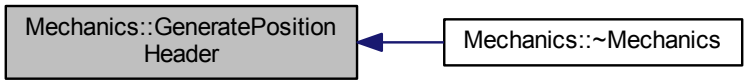
См. определение в файле Mechanics.cpp строка 94

```
95 {
96     if (isMoves)
97     {
98         std::stringstream positionFileName, positionFileNameCsv;
99         positionFileName << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport;
100         positionFileNameCsv << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport << ".csv";
101
102         std::ofstream newPositionFile(positionFileName.str());
103         std::ofstream newPositionFileCsv(positionFileNameCsv.str());
104
105         PrintLogoToTextFile(newPositionFile, positionFileName.str(), "Position of the
     airfoil " + W.getPassport().airfoilParams[numberInPassport].
    fileAirfoil);
106
107         PrintHeaderToTextFile(newPositionFile, "currentStep     currentTime     x
     y     phi     Vx     Vy     w");
108
109         newPositionFileCsv << "t,x,y,phi,Vx,Vy,w";
110
111         newPositionFile.close();
112         newPositionFile.clear();
113
114         newPositionFileCsv.close();
115         newPositionFileCsv.clear();
116     }
117 }//GeneratePositionHeader()
```

Граф вызовов:



Граф вызова функции:



10.12.3.7 GeneratePositionString()

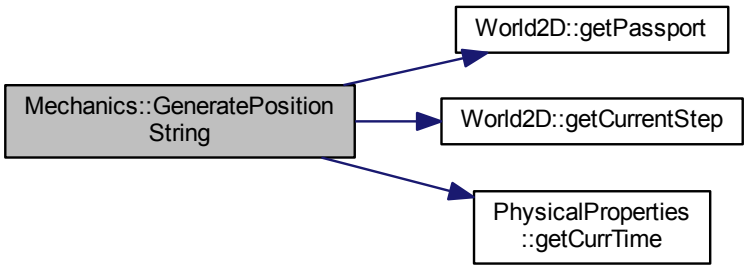
void Mechanics::GeneratePositionString ( ) [inherited]

Сохранение строки со статистикой в файл нагрузок

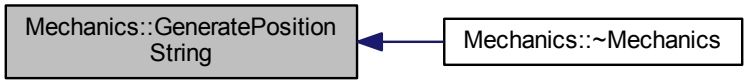
См. определение в файле Mechanics.cpp строка 138

```
139 {
140     if (isMoves)
141     {
142         std::stringstream positionFileName, positionFileNameCsv;
143         positionFileName << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport;
144         positionFileNameCsv << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport << ".csv";
145
146         std::ofstream forcesFile(positionFileName.str(), std::ios::app);
147         forcesFile << std::endl << W.getCurrentStep() << "   " <<
    W.getPassport().physicalProperties.getCurrTime() << "  " <<
    afl.rcm[0] << "   " << afl.rcm[1] << "  " << Phi << "    " << Vcm[0] << " " <<
    Vcm[1] << "  " << Wcm;
148         forcesFile.close();
149
150         std::ofstream forcesFileCsv(positionFileNameCsv.str(), std::ios::app);
151         forcesFileCsv << std::endl << W.getPassport().
    physicalProperties.getCurrTime() << "," << afl.
    rcm[0] << "," << afl.rcm[1] << "," << Phi << "," << Vcm[0] << "," <<
    Vcm[1] << "," << Wcm;
152         forcesFileCsv.close();
153     }
154 }
```

Граф вызовов:



Граф вызова функции:



10.12.3.8 GetHydroDynamForce()

void MechanicsRigidOscillMon::GetHydroDynamForce (
timePeriod & time) [virtual]

Вычисление гидродинамической силы, действующей на профиль

Аргументы

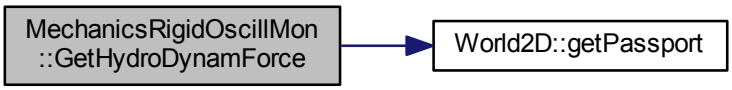
out	time	ссылка на промежуток времени — пару чисел (время начала и время конца операции)

Замещает Mechanics.

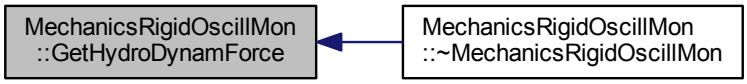
См. определение в файле MechanicsRigidOscillMon.cpp строка 45

```
46 {
47     time.first = omp_get_wtime();
48
49     const double& dt = W.getPassport().timeDiscretizationProperties
   .dt;
50
51     hydroDynamForce = { 0.0, 0.0 };
52     hydroDynamMoment = 0.0;
53
54     Point2D hDFGam = { 0.0, 0.0 };   //гидродинамические силы, обусловленные Gamma_k
55     Point2D hDFdelta = { 0.0, 0.0 }; //гидродинамические силы, обусловленные delta_k
56     double hDMdelta = 0.0;
57
58
59     Point2D deltaVstep = { 0.0, Vcm[1] - VcmOld[1] };
60
61     for (size_t i = 0; i < afl.np; ++i)
62     {
63
64         double deltaK = boundary.sheets.freeVortexSheet[i][0] *
   afl.len[i] - afl.gammaThrough[i] + deltaVstep * afl.tau[i] *
   afl.len[i];
65         Point2D rK = 0.5 * (afl.r[i + 1] + afl.r[i]) - afl.rcm;
66
67         hDFdelta += deltaK * Point2D({ -rK[1], rK[0] });
68         hDMdelta += 0.5 * deltaK * (rK * rK);
69     }
70
71     hydroDynamForce = hDFdelta * (1.0 / dt);
72     hydroDynamMoment = hDMdelta / dt;
73
74     time.second = omp_get_wtime();
75 }// GetHydroDynamForce(...)
```

Граф вызовов:



Граф вызова функции:



10.12.3.9 Move()

void MechanicsRigidOscillMon::Move ( ) [virtual]

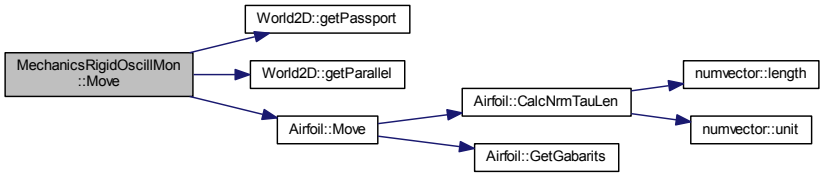
Перемещение профиля в соответствии с законом

Замещает Mechanics.

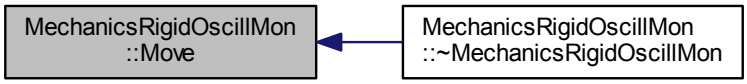
См. определение в файле MechanicsRigidOscillMon.cpp строка 101

```
{
    VcmOld = Vcm;

    Point2D dRcm = Vcm * W.getPassport().timeDiscretizationProperties.dt;

    MPI_Bcast(&dRcm, 1, Point2D::mpiPoint2D, 0, W.getParallel().commWork);

    Rcm += dRcm;
    afl.Move(dRcm);
}//Move()
```

Граф вызовов:



Граф вызова функции:



10.12.3.10 PositionOfAirfoilRcm()

Point2D MechanicsRigidOscillMon::PositionOfAirfoilRcm (
double currTime) [virtual]

Вычисление положения центра масс профиля

Аргументы

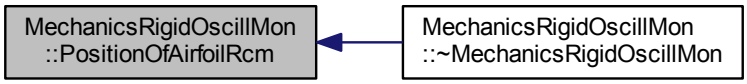
in	currTime	текущее время

Замещает Mechanics.

См. определение в файле MechanicsRigidOscillMon.cpp строка 85

```
{
    return Rcm;
    //return{ 0.0, 0.0 };
}//PositionOfAirfoilRcm(...)
```

Граф вызова функции:



10.12.3.11 ReadSpecificParametersFromDictionary()

void MechanicsRigidOscillMon::ReadSpecificParametersFromDictionary ( ) [virtual]

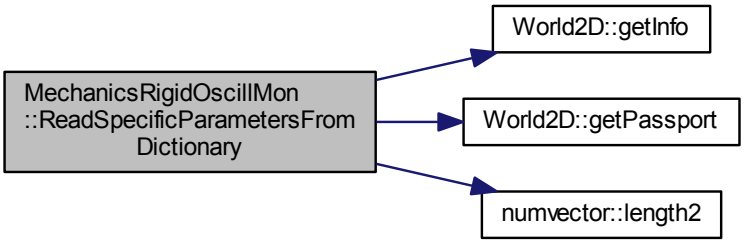
Чтение параметров конкретной механической системы

Замещает Mechanics.

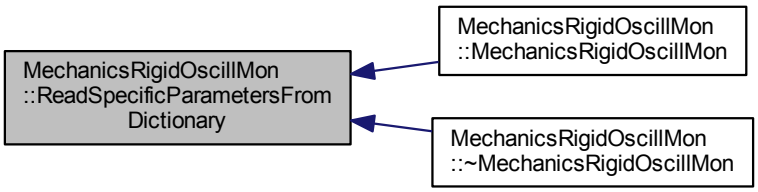
См. определение в файле MechanicsRigidOscillMon.cpp строка 113

```
{
    mechParamsParser->get("m", m);

    W.getInfo('i') << "mass m = " << m << std::endl;
    std::vector<double> sh;
    mechParamsParser->get("sh", sh);

    k =  m * 4.0 * PI * PI * sh[1] * sh[1] * W.getPassport().physicalProperties.vInf.length2();

    W.getInfo('i') << "rigidity k = " << k << std::endl;
}
```

Граф вызовов:



Граф вызова функции:



10.12.3.12 SolutionToMechanicalSystem()

void MechanicsRigidOscillMon::SolutionToMechanicalSystem (
Eigen::VectorXd & col) [virtual]

Извлечение из решения параметров механической системы

Аргументы

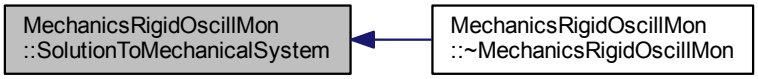
in	col	

Замещает Mechanics.

См. определение в файле MechanicsRigidOscillMon.cpp строка 158

```
{
    VcmOld = Vcm;

    Vcm[1] = col(0);

    for (size_t i = 0; i < afl.np + 1; ++i)
        afl.v[i] = Vcm;
}
```

Граф вызова функции:



10.12.3.13 VeloOfAirfoilPanels()

void MechanicsRigidOscillMon::VeloOfAirfoilPanels (
double currTime) [virtual]

Вычисление скоростей начал панелей

Аргументы

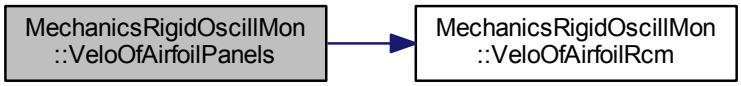
in	currTime	текущее время

Замещает Mechanics.

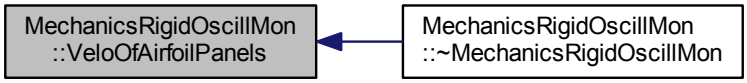
См. определение в файле MechanicsRigidOscillMon.cpp строка 92

```
{
    Point2D veloRcm = VeloOfAirfoilRcm(currTime);
    for (size_t i = 0; i < afl.v.size(); ++i)
        afl.v[i] = veloRcm;
    //afl.v[i] = { 0.0, 0.0 };
}//VeloOfAirfoilPanels(...)
```

Граф вызовов:



Граф вызова функции:



10.12.3.14 VeloOfAirfoilRcm()

Point2D MechanicsRigidOscillMon::VeloOfAirfoilRcm (
double currTime) [virtual]

Вычисление скорости центра масс профиля

Аргументы

in	currTime	текущее время

Замещает Mechanics.

См. определение в файле MechanicsRigidOscillMon.cpp строка 78

```
79 {
80     return Vcm;
81     //return{ 0.0, 0.0 };
82 }//VeloOfAirfoilRcm(...)
```

Граф вызова функции:

10.12.4 Данные класса

10.12.4.1 afl

Airfoil& Mechanics::afl [protected], [inherited]

Константная ссылка на профиль
инициализируется автоматом в конструкторе, при помощи const_cast

Предупреждения

использует const_cast для получения неконстантной ссылки

См. определение в файле Mechanics.h строка 83

10.12.4.2 b

const double MechanicsRigidOscillMon::b [private]

См. определение в файле MechanicsRigidOscillMon.h строка 67

10.12.4.3 boundary

const Boundary& Mechanics::boundary [protected], [inherited]

Константная ссылка на граничное условие
инициализируется автоматом в конструкторе

См. определение в файле Mechanics.h строка 87

10.12.4.4 degOfFreedom

const size_t Mechanics::degOfFreedom [inherited]

Количество степеней свободы

См. определение в файле Mechanics.h строка 125

10.12.4.5 hydroDynamForce

Point2D Mechanics::hydroDynamForce [inherited]

Вектор гидродинамической силы и момент, действующие на профиль

См. определение в файле Mechanics.h строка 128

10.12.4.6 hydroDynamMoment

double Mechanics::hydroDynamMoment [inherited]

См. определение в файле Mechanics.h строка 129

10.12.4.7 isDeform

const bool Mechanics::isDeform [inherited]

Переменная, отвечающая за то, деформируется профиль или нет

См. определение в файле Mechanics.h строка 119

10.12.4.8 isMoves

const bool Mechanics::isMoves [inherited]

Переменная, отвечающая за то, двигается профиль или нет

См. определение в файле Mechanics.h строка 116

10.12.4.9 isRotate

const bool Mechanics::isRotate [inherited]

Переменная, отвечающая за то, может профиль вращаться или нет

См. определение в файле Mechanics.h строка 122

10.12.4.10 k

double MechanicsRigidOscillMon::k [private]

См. определение в файле MechanicsRigidOscillMon.h строка 69

10.12.4.11 m

double MechanicsRigidOscillMon::m [private]

См. определение в файле MechanicsRigidOscillMon.h строка 66

10.12.4.12 mechParamsParser

std::unique_ptr<StreamParser> Mechanics::mechParamsParser [protected], [inherited]

Умный указатель на парсер параметров механической системы

См. определение в файле Mechanics.h строка 94

10.12.4.13 numberInPassport

const size_t Mechanics::numberInPassport [protected], [inherited]

Номер профиля в паспорте

См. определение в файле Mechanics.h строка 78

10.12.4.14 Phi

double Mechanics::Phi [protected], [inherited]

См. определение в файле Mechanics.h строка 106

10.12.4.15 Phi0

double Mechanics::Phi0 [protected], [inherited]

См. определение в файле Mechanics.h строка 100

10.12.4.16 PhiOld

double Mechanics::PhiOld [protected], [inherited]

См. определение в файле Mechanics.h строка 112

10.12.4.17 Rcm

Point2D Mechanics::Rcm [protected], [inherited]

Текущие положение профиля

См. определение в файле Mechanics.h строка 106

10.12.4.18 Rcm0

const Point2D Mechanics::Rcm0 [protected], [inherited]

Начальное положение профиля

См. определение в файле Mechanics.h строка 100

10.12.4.19 RcmOld

Point2D Mechanics::RcmOld [protected], [inherited]

Текущие положение профиля

См. определение в файле Mechanics.h строка 112

10.12.4.20 Vcm

Point2D Mechanics::Vcm [protected], [inherited]

Текущие скорость центра и угловая скорость

См. определение в файле Mechanics.h строка 103

10.12.4.21 Vcm0

const Point2D Mechanics::Vcm0 [protected], [inherited]

Начальная скорость центра и угловая скорость

См. определение в файле Mechanics.h строка 97

10.12.4.22 VcmOld

Point2D Mechanics::VcmOld [protected], [inherited]

Скорость и отклонение с предыдущего шага

См. определение в файле Mechanics.h строка 109

10.12.4.23 virtVortParams

const VortexesParams& Mechanics::virtVortParams [protected], [inherited]

Константная ссылка на структуру с параметрами виртуального вихревого слоя для профиля инициализируется автоматом в конструкторе

См. определение в файле Mechanics.h строка 91

10.12.4.24 W

const World2D& Mechanics::W [protected], [inherited]

Константная ссылка на решаемую задачу

См. определение в файле Mechanics.h строка 75

10.12.4.25 Wcm

double Mechanics::Wcm [protected], [inherited]

См. определение в файле Mechanics.h строка 103

10.12.4.26 Wcm0

const double Mechanics::Wcm0 [protected], [inherited]

См. определение в файле Mechanics.h строка 97

10.12.4.27 WcmOld

double Mechanics::WcmOld [protected], [inherited]

См. определение в файле Mechanics.h строка 109

Объявления и описания членов классов находятся в файлах:

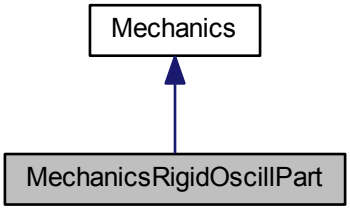
- Mechanics/MechanicsRigidOscillMon.h
- Mechanics/MechanicsRigidOscillMon.cpp

10.13 Класс MechanicsRigidOscillPart

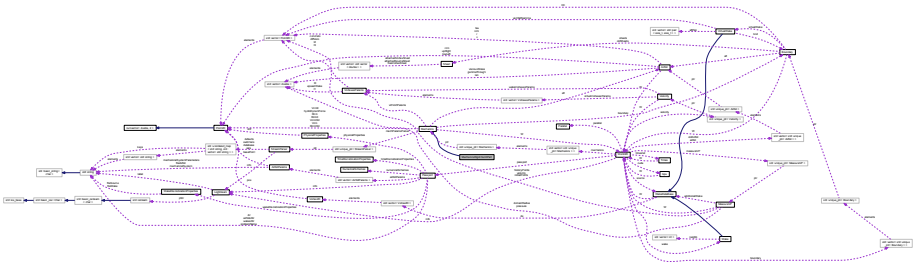
Класс, определяющий вид механической системы

#include <MechanicsRigidOscillPart.h>

Граф наследования:MechanicsRigidOscillPart:



Граф связей класса MechanicsRigidOscillPart:



Открытые члены

- MechanicsRigidOscillPart (const World2D &W_, size_t numberInPassport_)

  Конструктор
- ~MechanicsRigidOscillPart ()

  Деструктор
- virtual void GetHydroDynamForce (timePeriod &time)

  Вычисление гидродинамической силы, действующей на профиль
- virtual Point2D VeloOfAirfoilRcm (double currTime)

  Вычисление скорости центра масс профиля
- virtual Point2D PositionOfAirfoilRcm (double currTime)

  Вычисление положения центра масс профиля
- virtual void VeloOfAirfoilPanels (double currTime)

  Вычисление скоростей начал панелей
- virtual void ReadSpecificParametersFromDictionary ()

  Чтение параметров конкретной механической системы
- virtual void FillMechanicsRowsAndCross (Eigen::MatrixXd &row, Eigen::MatrixXd &cross)
- virtual void FillMechanicsRhs (std::vector< double > &rhs)

  Заполнение правых частей, соответствующих механической системе
- virtual void FillAtt (Eigen::MatrixXd &row, Eigen::MatrixXd &rhs)

  Заполнение правых частей (расщепленная схема) либо коэффициентов в матрице при скорости (монолитный подход), соответствующих присоединенным слоям
- virtual void SolutionToMechanicalSystem (Eigen::VectorXd &col)

  Извлечение из решения параметров механической системы
- virtual void Move ()

  Перемещение профиля в соответствии с законом
- void GenerateForcesHeader ()

  Генерация заголовка файла нагрузок
- void GeneratePositionHeader ()

  Генерация заголовка файла положения профиля
- void GenerateForcesString ()

  Сохранение строки со статистикой в файл нагрузок
- void GeneratePositionString ()

  Сохранение строки со статистикой в файл нагрузок

Открытые атрибуты

- const bool isMoves

  Переменная, отвечающая за то, двигается профиль или нет
- const bool isDeform

  Переменная, отвечающая за то, деформируется профиль или нет
- const bool isRotate

  Переменная, отвечающая за то, может профиль вращаться или нет
- const size_t degOfFreedom

  Количество степеней свободы
- Point2D hydroDynamForce

  Вектор гидродинамической силы и момент, действующие на профиль
- double hydroDynamMoment

Защищенные данные

- const World2D & W

  Константная ссылка на решаемую задачу
- const size_t numberInPassport

  Номер профиля в паспорте
- Airfoil & afl
- const Boundary & boundary
- const VortexesParams & virtVortParams
- std::unique_ptr< StreamParser > mechParamsParser

  Умный указатель на парсер параметров механической системы
- const Point2D Vcm0

  Начальная скорость центра и угловая скорость
- const double Wcm0
- const Point2D Rcm0

  Начальное положение профиля
- double Phi0
- Point2D Vcm

  Текущие скорость центра и угловая скорость
- double Wcm
- Point2D Rcm

  Текущие положение профиля
- double Phi
- Point2D VcmOld

  Скорость и отклонение с предыдущего шага
- double WcmOld
- Point2D RcmOld

  Текущие положение профиля
- double PhiOld

Закрытые данные

- const double u0

  начальная скорость профиля
- const double y0

  начальное отклонение профиля
- double u

  текущая скорость профиля
- double y

  текущее отклонение профиля
- double uOld

  скорость профиля с предыдущего шага
- double yOld

  отклонение профиля с предыдущего шага
- double m

  масса профиля
- const double b

  параметр демпфирования механической системы
- double k

  параметр жесткости механической системы

10.13.1 Подробное описание

Класс, определяющий вид механической системы

Упруго закрепленное тело, метод расщепления

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле MechanicsRigidOscillPart.h строка 59

10.13.2 Конструктор(ы)

10.13.2.1 MechanicsRigidOscillPart()

```
MechanicsRigidOscillPart::MechanicsRigidOscillPart (
            const World2D & W_,
            size_t numberInPassport_ )   [inline]
```

Конструктор

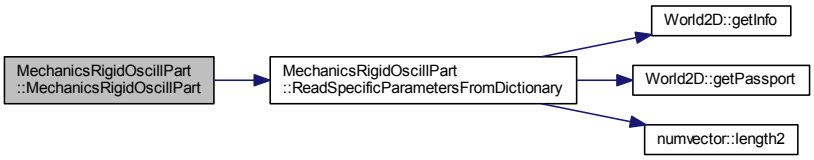
Аргументы

in	W_	константная ссылка на решаемую задачу
in	numberIn↩ Passport_	номер профиля в паспорте задачи

См. определение в файле MechanicsRigidOscillPart.h строка 97

```
98      : Mechanics(W_, numberInPassport_, 0, true, false, false, { 0.0, 0.0 }, { 0.0, 0.0 }, 0.0,
    0.0), u0(0.0), y0(0.0), b(0.0*0.731)
99    {
100       u = u0;
101       y = y0;
102       uOld = u0;
103       yOld = y0;
104       ReadSpecificParametersFromDictionary();
105   };
```

Граф вызовов:



10.13.2.2 ~MechanicsRigidOscillPart()

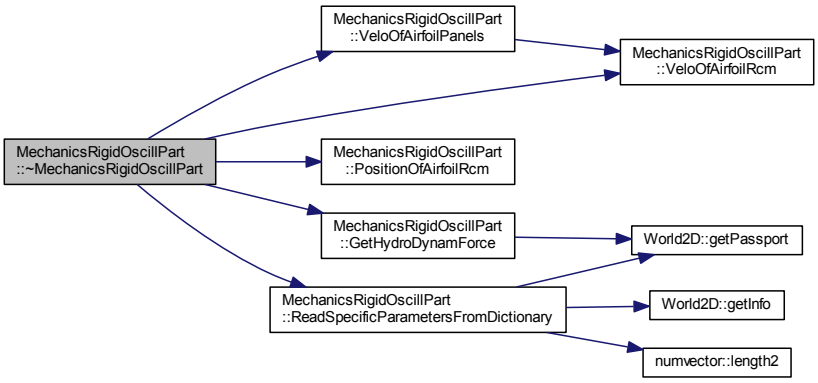
MechanicsRigidOscillPart::~MechanicsRigidOscillPart ( ) [inline]

Деструктор

См. определение в файле MechanicsRigidOscillPart.h строка 108

```
108 {};
```

Граф вызовов:



10.13.3 Методы

10.13.3.1 FillAtt()

void MechanicsRigidOscillPart::FillAtt (
Eigen::MatrixXd & row,
Eigen::MatrixXd & rhs) [virtual]

Заполнение правых частей (расщепленная схема) либо коэффициентов в матрице при скорости (монолитный подход), соответствующих присоединенным слоям

Аргументы

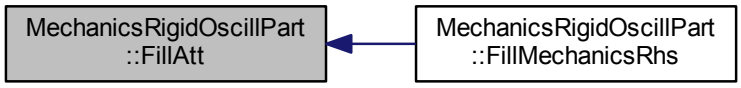
in	row	строки (количество строк = количество степеней свободы, количество элементов в строках = количество неизвестных в локальной матрице)
in	rhs	вектор правой части для механики одного профиля (количество элементов = количеству степеней свободы)

Замещает Mechanics.

См. определение в файле MechanicsRigidOscillPart.cpp строка 148

```
149 {
150     for (int i = 0; i < afl.np; ++i)
151         rhs(i, 0) = /*12*/ afl.v[i] * afl.tau[i];
152 }
```

Граф вызова функции:



10.13.3.2 FillMechanicsRhs()

virtual void MechanicsRigidOscillPart::FillMechanicsRhs (
std::vector< double > & rhs) [inline], [virtual]

Заполнение правых частей, соответствующих механической системе

Аргументы

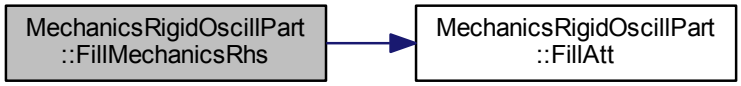
in	rhs	вектор правой части для механики одного профиля (количество элементов = количеству степеней свободы)

Замещает Mechanics.

См. определение в файле MechanicsRigidOscillPart.h строка 120

```
120 {};
```

Граф вызовов:



10.13.3.3 FillMechanicsRowsAndCross()

virtual void MechanicsRigidOscillPart::FillMechanicsRowsAndCross (
Eigen::MatrixXd & row,
Eigen::MatrixXd & cross) [inline], [virtual]

Необходимо сделать реализовать — не понял

Замещает Mechanics.

См. определение в файле MechanicsRigidOscillPart.h строка 119

```
119 {};
```

10.13.3.4 GenerateForcesHeader()

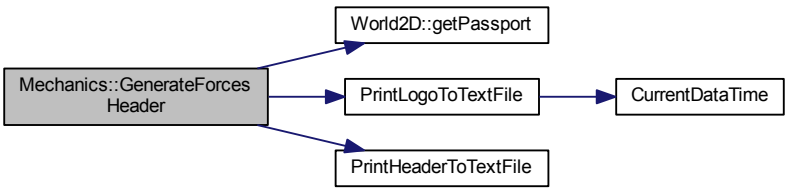
void Mechanics::GenerateForcesHeader ( ) [inherited]

Генерация заголовка файла нагрузок

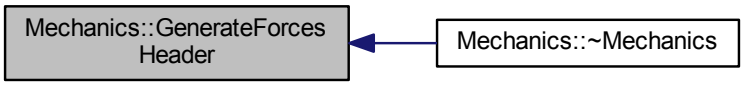
См. определение в файле Mechanics.cpp строка 69

```
70 {
71     std::stringstream forceFileName, forceFileNameCsv;
72     forceFileName << W.getPassport().dir << "forces-airfoil-" <<
    numberInPassport;
73     forceFileNameCsv << W.getPassport().dir << "forces-airfoil-" <<
    numberInPassport << ".csv";
74
75     std::ofstream newForcesFile(forceFileName.str());
76     std::ofstream newForcesFileCsv(forceFileNameCsv.str());
77
78     PrintLogoToTextFile(newForcesFile, forceFileName.str(), "Hydrodynamic loads for the
     airfoil " + W.getPassport().airfoilParams[
    numberInPassport].fileAirfoil);
79
80     PrintHeaderToTextFile(newForcesFile, "currentStep     currentTime     Fx     Fy
     Mz");
81
82     newForcesFileCsv << "t,Fx,Fy,Mz";
83
84     newForcesFile.close();
85     newForcesFile.clear();
86
87     newForcesFileCsv.close();
88     newForcesFileCsv.clear();
89
90 }//GenerateForcesHeader()
```

Граф вызовов:



Граф вызова функции:



10.13.3.5 GenerateForcesString()

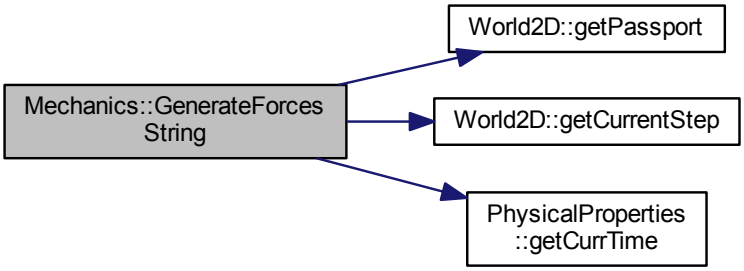
void Mechanics::GenerateForcesString ( ) [inherited]

Сохранение строки со статистикой в файл нагрузок

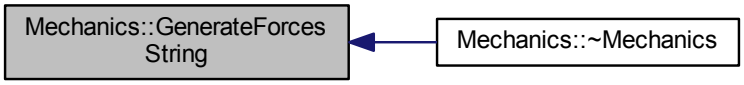
См. определение в файле Mechanics.cpp строка 121

```
{
    std::stringstream forceFileName, forceFileNameCsv;
    forceFileName << W.getPassport().dir << "forces-airfoil-" << numberInPassport;
    forceFileNameCsv << W.getPassport().dir << "forces-airfoil-" << numberInPassport << ".csv";

    std::ofstream forcesFile(forceFileName.str(), std::ios::app);
    forcesFile << std::endl << W.getCurrentStep() << "   " << W.getPassport().physicalProperties.getCurrTime() << " " << hydroDynamForce[0] << "  " << hydroDynamForce[1] << " " << hydroDynamMoment;
    forcesFile.close();

    std::ofstream forcesFileCsv(forceFileNameCsv.str(), std::ios::app);
    forcesFileCsv << std::endl << W.getPassport().physicalProperties.getCurrTime() << "," << hydroDynamForce[0] << "," << hydroDynamForce[1] << "," << hydroDynamMoment;
    forcesFileCsv.close();
}//GenerateForcesString()
```

Граф вызовов:



Граф вызова функции:



10.13.3.6 GeneratePositionHeader()

void Mechanics::GeneratePositionHeader ( ) [inherited]

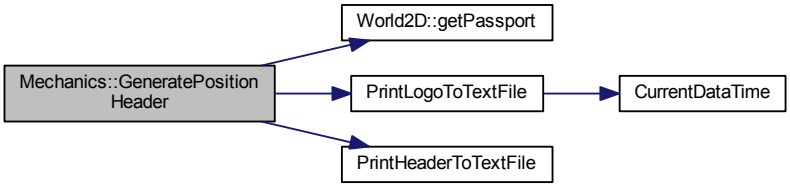
Генерация заголовка файла положения профиля

См. определение в файле Mechanics.cpp строка 94

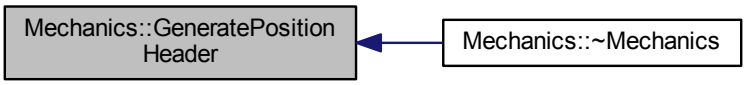
```
95 {
96     if (isMoves)
97     {
98         std::stringstream positionFileName, positionFileNameCsv;
99         positionFileName << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport;
100         positionFileNameCsv << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport << ".csv";
101
102         std::ofstream newPositionFile(positionFileName.str());
103         std::ofstream newPositionFileCsv(positionFileNameCsv.str());
104
105         PrintLogoToTextFile(newPositionFile, positionFileName.str(), "Position of the
     airfoil " + W.getPassport().airfoilParams[numberInPassport].
    fileAirfoil);
106
107         PrintHeaderToTextFile(newPositionFile, "currentStep     currentTime     x
     y     phi     Vx     Vy     w");
108
109         newPositionFileCsv << "t,x,y,phi,Vx,Vy,w";
```

```

        newPositionFile.close();
        newPositionFile.clear();

        newPositionFileCsv.close();
        newPositionFileCsv.clear();
    }
}//GeneratePositionHeader()
```

Граф вызовов:



Граф вызова функции:



10.13.3.7 GeneratePositionString()

void Mechanics::GeneratePositionString ( ) [inherited]

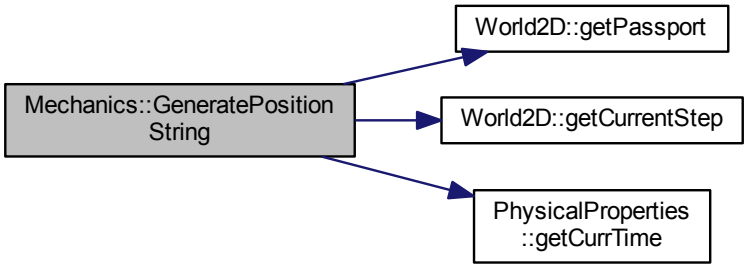
Сохранение строки со статистикой в файл нагрузок

См. определение в файле Mechanics.cpp строка 138

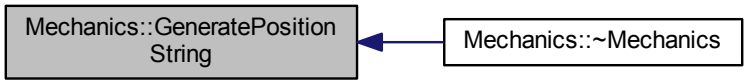
```
{
    if (isMoves)
    {
        std::stringstream positionFileName, positionFileNameCsv;
        positionFileName << W.getPassport().dir << "position-airfoil-" << numberInPassport;
        positionFileNameCsv << W.getPassport().dir << "position-airfoil-" << numberInPassport << ".csv";

        std::ofstream forcesFile(positionFileName.str(), std::ios::app);
        forcesFile << std::endl << W.getCurrentStep() << "   " << W.getPassport().physicalProperties.getCurrTime() << "  " << afl.rcm[0] << "   " << afl.rcm[1] << "  " << Phi << "    " << Vcm[0] << " " << Vcm[1] << "  " << Wcm;
        forcesFile.close();
```

```

        std::ofstream forcesFileCsv(positionFileNameCsv.str(), std::ios::app);
        forcesFileCsv << std::endl << W.getPassport().
physicalProperties.getCurrTime() << "," << afl.
rcm[0] << "," << afl.rcm[1] << "," << Phi << "," << Vcm[0] << "," <<
Vcm[1] << "," << Wcm;
        forcesFileCsv.close();
    }
}
```

Граф вызовов:



Граф вызова функции:



10.13.3.8 GetHydroDynamForce()

void MechanicsRigidOscillPart::GetHydroDynamForce (
timePeriod & time) [virtual]

Вычисление гидродинамической силы, действующей на профиль

Аргументы

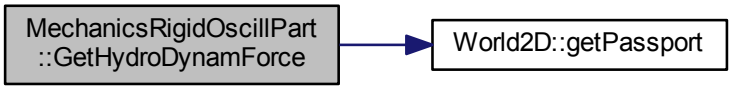
out	time	ссылка на промежуток времени — пару чисел (время начала и время конца операции)

Замещает Mechanics.

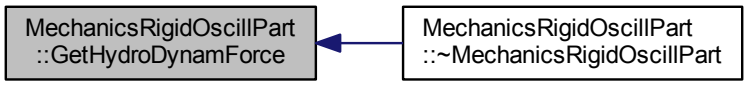
См. определение в файле MechanicsRigidOscillPart.cpp строка 45

```
46 {
47     time.first = omp_get_wtime();
48
49     const double& dt = W.getPassport().timeDiscretizationProperties
   .dt;
50
51     hydroDynamForce = { 0.0, 0.0 };
52     hydroDynamMoment = 0.0;
53
54     Point2D hDFGam = { 0.0, 0.0 };   //гидродинамические силы, обусловленные Gamma_k
55     Point2D hDFdelta = { 0.0, 0.0 }; //гидродинамические силы, обусловленные delta_k
56     double hDMdelta = 0.0;
57
58
59     Point2D deltaVstep = { 0.0, u - uOld };
60
61     for (size_t i = 0; i < afl.np; ++i)
62     {
63
64         double deltaK = boundary.sheets.freeVortexSheet[i][0] *
   afl.len[i] - afl.gammaThrough[i] + deltaVstep * afl.tau[i] *
   afl.len[i];
65         Point2D rK = 0.5 * (afl.r[i + 1] + afl.r[i]) - afl.rcm;
66
67         hDFdelta += deltaK * Point2D({ -rK[1], rK[0] });
68         hDMdelta += 0.5 * deltaK * (rK * rK);
69     }
70
71     hydroDynamForce = hDFdelta * (1.0 / dt);
72     hydroDynamMoment = hDMdelta / dt;
73
74     time.second = omp_get_wtime();
75 }// GetHydroDynamForce(...)
```

Граф вызовов:



Граф вызова функции:



10.13.3.9 Move()

void MechanicsRigidOscillPart::Move ( ) [virtual]

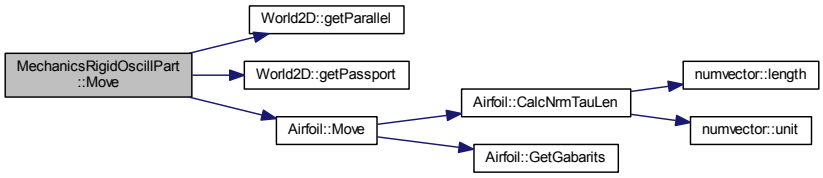
Перемещение профиля в соответствии с законом

Замещает Mechanics.

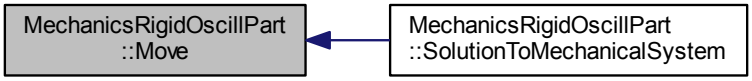
См. определение в файле MechanicsRigidOscillPart.cpp строка 101

```
102 {
103     uOld = u;
104     yOld = y;
105
106     double dy, du;
107
108     //W.getInfo('t') << "k = " << k << std::endl;
109
110     if (W.getParallel().myidWork == 0)
111     {
112         double dt = W.getPassport().timeDiscretizationProperties.
    dt;
113         numvector<double, 2> kk[4];
114         kk[0] = { u, (hydroDynamForce[1] - b*u - k*y) / m };
115         kk[1] = { u + 0.5*dt*kk[0][1], (hydroDynamForce[1] - b*(
    u + 0.5*dt*kk[0][1]) - k*(y + 0.5*dt*kk[0][0])) / m };
116         kk[2] = { u + 0.5*dt*kk[1][1], (hydroDynamForce[1] - b*(
    u + 0.5*dt*kk[1][1]) - k*(y + 0.5*dt*kk[1][0])) / m };
117         kk[3] = { u + dt*kk[2][1], (hydroDynamForce[1] - b*(u + dt*kk[2][1]) -
    k*(y + dt*kk[2][0])) / m };
118
119         dy = dt * (kk[0][0] + 2. * kk[1][0] + 2. * kk[2][0] + kk[3][0]) / 6.0;
120         du = dt * (kk[0][1] + 2. * kk[1][1] + 2. * kk[2][1] + kk[3][1]) / 6.0;
121     }
122
123     MPI_Bcast(&dy, 1, MPI_DOUBLE, 0, W.getParallel().commWork);
124     MPI_Bcast(&du, 1, MPI_DOUBLE, 0, W.getParallel().commWork);
125
126     afl.Move({ 0.0, dy });
127     Vcm[1] += du;
128
129     y += dy;
130     u += du;
131 }//Move()
```

Граф вызовов:



Граф вызова функции:



10.13.3.10 PositionOfAirfoilRcm()

Point2D MechanicsRigidOscillPart::PositionOfAirfoilRcm (
double currTime) [virtual]

Вычисление положения центра масс профиля

Аргументы

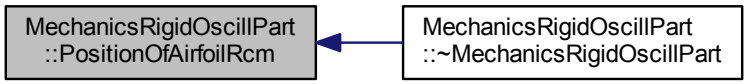
in	currTime	текущее время

Замещает Mechanics.

См. определение в файле MechanicsRigidOscillPart.cpp строка 85

```
86 {
87     return{ 0.0, y };
88     //return{ 0.0, 0.0 };
89 }//PositionOfAirfoilRcm(...)
```

Граф вызова функции:



10.13.3.11 ReadSpecificParametersFromDictionary()

void MechanicsRigidOscillPart::ReadSpecificParametersFromDictionary ( ) [virtual]

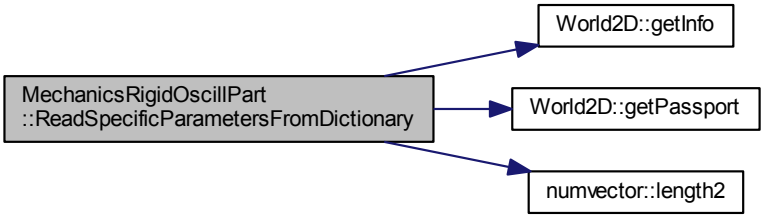
Чтение параметров конкретной механической системы

Замещает Mechanics.

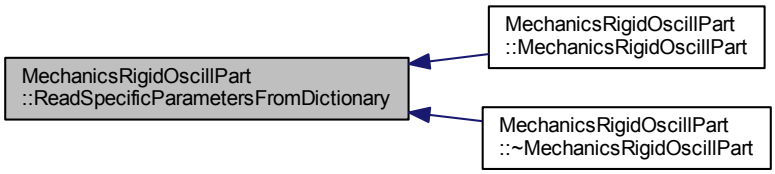
См. определение в файле MechanicsRigidOscillPart.cpp строка 134

```
135 {
136     mechParamsParser->get("m", m);
137
138     W.getInfo('i') << "mass " << "m = " << m << std::endl;
139
140     std::vector<double> sh;
141     mechParamsParser->get("sh", sh);
142
143     k = m * 4.0 * PI * PI * sh[1] * sh[1] * W.getPassport().
    physicalProperties.vInf.length2();
144
145     W.getInfo('i') << "rigidity k = " << k << std::endl;
146 }
```

Граф вызовов:



Граф вызова функции:



10.13.3.12 SolutionToMechanicalSystem()

virtual void MechanicsRigidOscillPart::SolutionToMechanicalSystem (
Eigen::VectorXd & col) [inline], [virtual]

Извлечение из решения параметров механической системы

Аргументы

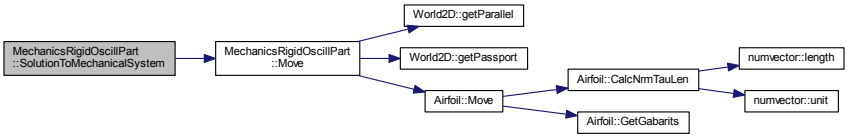
in	col	

Замещает Mechanics.

См. определение в файле MechanicsRigidOscillPart.h строка 122

122 {};

Граф вызовов:



10.13.3.13 VeloOfAirfoilPanels()

void MechanicsRigidOscillPart::VeloOfAirfoilPanels (
double currTime) [virtual]

Вычисление скоростей начал панелей

Аргументы

in	currTime	текущее время

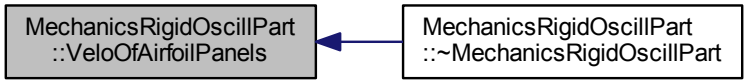
Замещает Mechanics.

См. определение в файле MechanicsRigidOscillPart.cpp строка 92

```
{
    Point2D veloRcm = VeloOfAirfoilRcm(currTime);
    for (size_t i = 0; i < afl.v.size(); ++i)
        afl.v[i] = veloRcm;
        //afl.v[i] = { 0.0, 0.0 };
}//VeloOfAirfoilPanels(...)
```

Граф вызовов:

Граф вызова функции:



10.13.3.14 VeloOfAirfoilRcm()

Point2D MechanicsRigidOscillPart::VeloOfAirfoilRcm (
double currTime) [virtual]

Вычисление скорости центра масс профиля

Аргументы

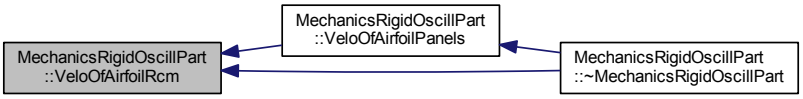
in	currTime	текущее время

Замещает Mechanics.

См. определение в файле MechanicsRigidOscillPart.cpp строка 78

```
79 {
80     return {0.0, u};
81     //return{ 0.0, 0.0 };
82 }//VeloOfAirfoilRcm(...)
```

Граф вызова функции:



10.13.4 Данные класса

10.13.4.1 afl

Airfoil& Mechanics::afl [protected], [inherited]

Константная ссылка на профиль
инициализируется автоматом в конструкторе, при помощи const_cast

Предупреждения

использует const_cast для получения неконстантной ссылки

См. определение в файле Mechanics.h строка 83

10.13.4.2 b

const double MechanicsRigidOscillPart::b [private]

параметр демпфирования механической системы

См. определение в файле MechanicsRigidOscillPart.h строка 86

10.13.4.3 boundary

const Boundary& Mechanics::boundary [protected], [inherited]

Константная ссылка на граничное условие
инициализируется автоматом в конструкторе

См. определение в файле Mechanics.h строка 87

10.13.4.4 degOfFreedom

const size_t Mechanics::degOfFreedom [inherited]

Количество степеней свободы

См. определение в файле Mechanics.h строка 125

10.13.4.5 hydroDynamForce

Point2D Mechanics::hydroDynamForce [inherited]

Вектор гидродинамической силы и момент, действующие на профиль

См. определение в файле Mechanics.h строка 128

10.13.4.6 hydroDynamMoment

double Mechanics::hydroDynamMoment [inherited]

См. определение в файле Mechanics.h строка 129

10.13.4.7 isDeform

const bool Mechanics::isDeform [inherited]

Переменная, отвечающая за то, деформируется профиль или нет

См. определение в файле Mechanics.h строка 119

10.13.4.8 isMoves

const bool Mechanics::isMoves [inherited]

Переменная, отвечающая за то, двигается профиль или нет

См. определение в файле Mechanics.h строка 116

10.13.4.9 isRotate

const bool Mechanics::isRotate [inherited]

Переменная, отвечающая за то, может профиль вращаться или нет

См. определение в файле Mechanics.h строка 122

10.13.4.10 k

double MechanicsRigidOscillPart::k [private]

параметр жесткости механической системы

См. определение в файле MechanicsRigidOscillPart.h строка 89

10.13.4.11 m

double MechanicsRigidOscillPart::m [private]

масса профиля

См. определение в файле MechanicsRigidOscillPart.h строка 83

10.13.4.12 mechParamsParser

std::unique_ptr<StreamParser> Mechanics::mechParamsParser [protected], [inherited]

Умный указатель на парсер параметров механической системы

См. определение в файле Mechanics.h строка 94

10.13.4.13 numberInPassport

const size_t Mechanics::numberInPassport [protected], [inherited]

Номер профиля в паспорте

См. определение в файле Mechanics.h строка 78

10.13.4.14 Phi

double Mechanics::Phi [protected], [inherited]

См. определение в файле Mechanics.h строка 106

10.13.4.15 Phi0

double Mechanics::Phi0 [protected], [inherited]

См. определение в файле Mechanics.h строка 100

10.13.4.16 PhiOld

double Mechanics::PhiOld [protected], [inherited]

См. определение в файле Mechanics.h строка 112

10.13.4.17 Rcm

Point2D Mechanics::Rcm [protected], [inherited]

Текущие положение профиля

См. определение в файле Mechanics.h строка 106

10.13.4.18 Rcm0

const Point2D Mechanics::Rcm0 [protected], [inherited]

Начальное положение профиля

См. определение в файле Mechanics.h строка 100

10.13.4.19 RcmOld

Point2D Mechanics::RcmOld [protected], [inherited]

Текущие положение профиля

См. определение в файле Mechanics.h строка 112

10.13.4.20 u

double MechanicsRigidOscillPart::u [private]

текущая скорость профиля

См. определение в файле MechanicsRigidOscillPart.h строка 71

10.13.4.21 u0

const double MechanicsRigidOscillPart::u0 [private]

начальная скорость профиля

См. определение в файле MechanicsRigidOscillPart.h строка 65

10.13.4.22 uOld

double MechanicsRigidOscillPart::uOld [private]

скорость профиля с предыдущего шага

См. определение в файле MechanicsRigidOscillPart.h строка 77

10.13.4.23 Vcm

Point2D Mechanics::Vcm [protected], [inherited]

Текущие скорость центра и угловая скорость

См. определение в файле Mechanics.h строка 103

10.13.4.24 Vcm0

const Point2D Mechanics::Vcm0 [protected], [inherited]

Начальная скорость центра и угловая скорость

См. определение в файле Mechanics.h строка 97

10.13.4.25 VcmOld

Point2D Mechanics::VcmOld [protected], [inherited]

Скорость и отклонение с предыдущего шага

См. определение в файле Mechanics.h строка 109

10.13.4.26 virtVortParams

const VortexesParams& Mechanics::virtVortParams [protected], [inherited]

Константная ссылка на структуру с параметрами виртуального вихревого слоя для профиля инициализируется автоматом в конструкторе

См. определение в файле Mechanics.h строка 91

10.13.4.27 W

const World2D& Mechanics::W [protected], [inherited]

Константная ссылка на решаемую задачу

См. определение в файле Mechanics.h строка 75

10.13.4.28 Wcm

double Mechanics::Wcm [protected], [inherited]

См. определение в файле Mechanics.h строка 103

10.13.4.29 Wcm0

const double Mechanics::Wcm0 [protected], [inherited]

См. определение в файле Mechanics.h строка 97

10.13.4.30 WcmOld

double Mechanics::WcmOld [protected], [inherited]

См. определение в файле Mechanics.h строка 109

10.13.4.31 y

double MechanicsRigidOscillPart::y [private]

текущее отклонение профиля

См. определение в файле MechanicsRigidOscillPart.h строка 74

10.13.4.32 y0

const double MechanicsRigidOscillPart::y0 [private]

начальное отклонение профиля

См. определение в файле MechanicsRigidOscillPart.h строка 68

10.13.4.33 yOld

double MechanicsRigidOscillPart::yOld [private]

отклонение профиля с предыдущего шага

См. определение в файле MechanicsRigidOscillPart.h строка 80

Объявления и описания членов классов находятся в файлах:

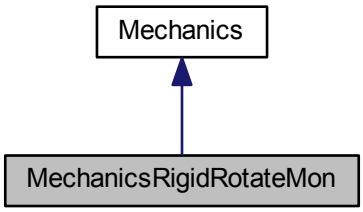
- Mechanics/MechanicsRigidOscillPart.h
- Mechanics/MechanicsRigidOscillPart.cpp

10.14 Класс MechanicsRigidRotateMon

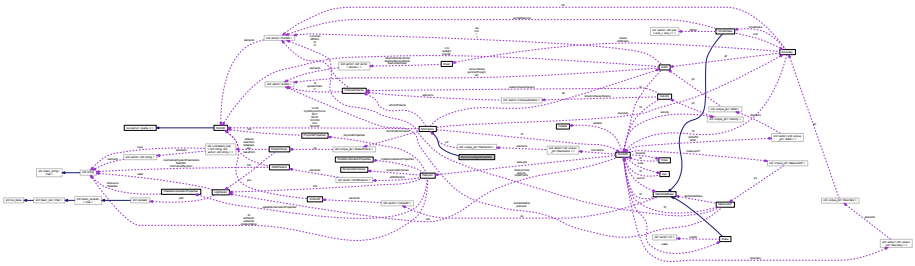
Класс, определяющий вид механической системы

#include <MechanicsRigidRotateMon.h>

Граф наследования:MechanicsRigidRotateMon:



Граф связей класса MechanicsRigidRotateMon:



Открытые члены

- MechanicsRigidRotateMon (const World2D &W_, size_t numberInPassport_)

  Конструктор
- ~MechanicsRigidRotateMon ()

  Деструктор
- virtual void GetHydroDynamForce (timePeriod &time)

  Вычисление гидродинамической силы, действующей на профиль
- virtual Point2D VeloOfAirfoilRcm (double currTime)

  Вычисление скорости центра масс профиля
- virtual Point2D PositionOfAirfoilRcm (double currTime)

  Вычисление положения центра масс профиля
- virtual void VeloOfAirfoilPanels (double currTime)

  Вычисление скоростей начал панелей
- virtual void ReadSpecificParametersFromDictionary ()

  Чтение параметров конкретной механической системы
- virtual void FillMechanicsRowsAndCross (Eigen::MatrixXd &row, Eigen::MatrixXd &cross)

  Заполнение строк в матрице, соответствующих механической системе. А также пересечений строк и столбцов, соответствующих мех. системе
- virtual void FillMechanicsRhs (std::vector< double > &rhs)

  Заполнение правых частей, соответствующих механической системе
- virtual void FillAtt (Eigen::MatrixXd &row, Eigen::MatrixXd &rhs)

  Заполнение правых частей (расщепленная схема) либо коэффициентов в матрице при скорости (монолитный подход), соответствующих присоединенным слоям
- virtual void SolutionToMechanicalSystem (Eigen::VectorXd &col)

  Извлечение из решения параметров механической системы
- virtual void Move ()

  Перемещение профиля в соответствии с законом
- void GenerateForcesHeader ()

  Генерация заголовка файла нагрузок
- void GeneratePositionHeader ()

  Генерация заголовка файла положения профиля
- void GenerateForcesString ()

  Сохранение строки со статистикой в файл нагрузок
- void GeneratePositionString ()

  Сохранение строки со статистикой в файл нагрузок

Открытые атрибуты

- const bool isMoves

  Переменная, отвечающая за то, двигается профиль или нет
- const bool isDeform

  Переменная, отвечающая за то, деформируется профиль или нет
- const bool isRotate

  Переменная, отвечающая за то, может профиль вращаться или нет
- const size_t degOfFreedom

  Количество степеней свободы
- Point2D hydroDynamForce

  Вектор гидродинамической силы и момент, действующие на профиль
- double hydroDynamMoment

Защищенные данные

- const World2D & W

  Константная ссылка на решаемую задачу
- const size_t numberInPassport

  Номер профиля в паспорте
- Airfoil & afl
- const Boundary & boundary
- const VortexesParams & virtVortParams
- std::unique_ptr< StreamParser > mechParamsParser

  Умный указатель на парсер параметров механической системы
- const Point2D Vcm0

  Начальная скорость центра и угловая скорость
- const double Wcm0
- const Point2D Rcm0

  Начальное положение профиля
- double Phi0
- Point2D Vcm

  Текущие скорость центра и угловая скорость
- double Wcm
- Point2D Rcm

  Текущие положение профиля
- double Phi
- Point2D VcmOld

  Скорость и отклонение с предыдущего шага
- double WcmOld
- Point2D RcmOld

  Текущие положение профиля
- double PhiOld

Закрытые члены

- double Alpha (const Point2D &p, const Point2D &s) const

  Вспомогательная функция вычисления угла между векторами
- double Lambda (const Point2D &p, const Point2D &s) const

  Вспомогательная функция вычисления логарифма отношения норм векторов
- double GetMz (double t)

Закрытые данные

- double J
- double b
- double k
- double tRotateAccel

  время, в течение которого ротор крутится без нагрузки
- double tMomentAccel

  время, в течение которого постепенно увеличивается момент, которым нагружается ротор
- double Mz

  Момент, который снимается с ротора

10.14.1 Подробное описание

Класс, определяющий вид механической системы

Жесткое тело, движущееся по заданному закону

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле MechanicsRigidRotateMon.h строка 59

10.14.2 Конструктор(ы)

10.14.2.1 MechanicsRigidRotateMon()

```
MechanicsRigidRotateMon::MechanicsRigidRotateMon (
            const World2D & W_,
            size_t numberInPassport_ )   [inline]
```

Конструктор

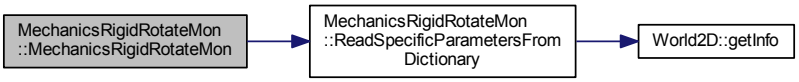
Аргументы

in	W_	константная ссылка на решаемую задачу
in	numberIn↩ Passport_	номер профиля в паспорте задачи

См. определение в файле MechanicsRigidRotateMon.h строка 113

```
114         : Mechanics(W_, numberInPassport_, 1, true, false, true, { 0.0, 0.0 }, {0.0, 0.0}, 0.0, 0.
    0), b(0.0 * 0.731)
115     {
116         ReadSpecificParametersFromDictionary();
117     };
```

Граф вызовов:



10.14.2.2 ~MechanicsRigidRotateMon()

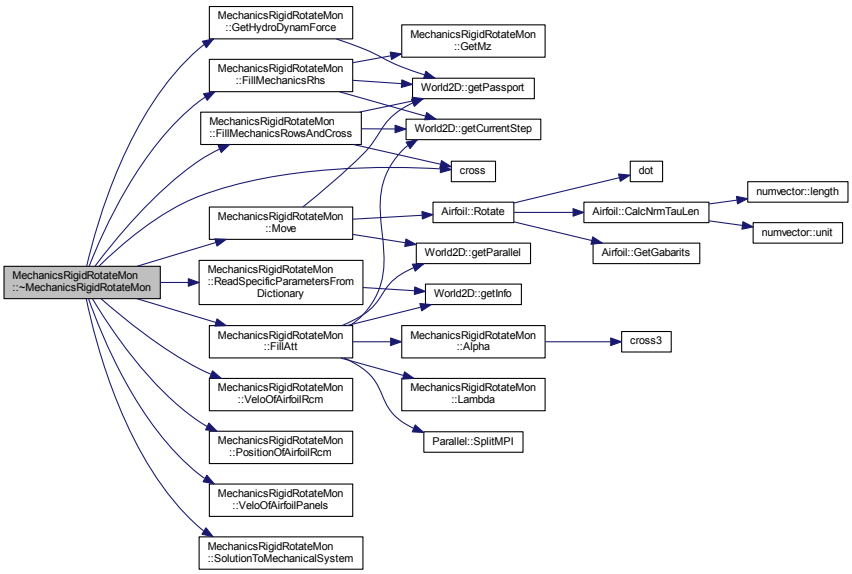
MechanicsRigidRotateMon::~MechanicsRigidRotateMon ( ) [inline]

Деструктор

См. определение в файле MechanicsRigidRotateMon.h строка 120

```
120 {};
```

Граф вызовов:



10.14.3 Методы

10.14.3.1 Alpha()

double MechanicsRigidRotateMon::Alpha (
const Point2D & p,
const Point2D & s) const [inline], [private]

Вспомогательная функция вычисления угла между векторами

Аргументы

in	p	константная ссылка на первый вектор
in	s	константная ссылка на второй вектор

Возвращает

угол между векторами в диапазоне $(-\pi; \pi]$

См. определение в файле MechanicsRigidRotateMon.h строка 82

```
83    {
84        return atan2(cross3(p, s), p*s);
85    }
```

Граф вызовов:

Граф вызова функции:

10.14.3.2 FillAtt()

```
void MechanicsRigidRotateMon::FillAtt (
            Eigen::MatrixXd & row,
            Eigen::MatrixXd & rhs )   [virtual]
```

Заполнение правых частей (расщепленная схема) либо коэффициентов в матрице при скорости (монолитный подход), соответствующих присоединенным слоям

Аргументы

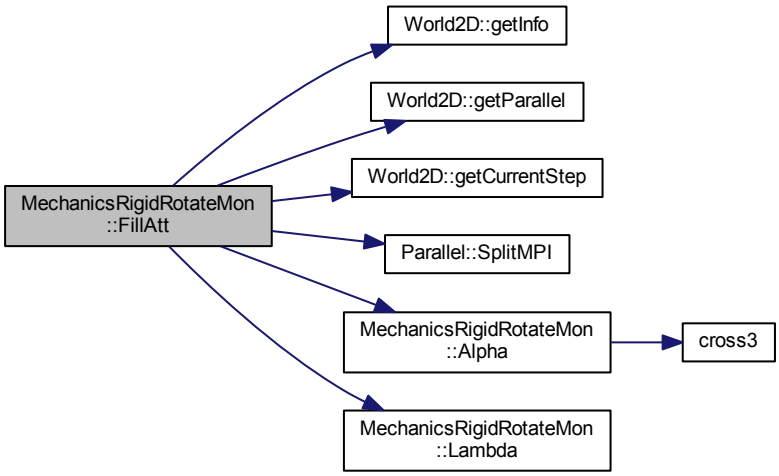
in	row	строки (количество строк = количество степеней свободы, количество элементов в строках = количество неизвестных в локальной матрице)
in	rhs	вектор правой части для механики одного профиля (количество элементов = количеству степеней свободы)

Замещает Mechanics.

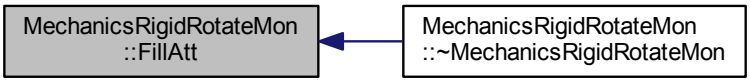
См. определение в файле MechanicsRigidRotateMon.cpp строка 204

```
205 {
206     W.getInfo('t') << "FillAtt: " << W.getParallel().
    myidWork << std::endl;
207
208     if (W.getCurrentStep() == 0)
209     {
210         Point2D riC;
211
212
213         std::vector<double> colMPI;
214         colMPI.resize(col.rows(), 0.0);
215
216         int id = W.getParallel().myidWork;
217
218         parProp par = W.getParallel().SplitMPI(col.rows());
219
220         std::vector<double> locColMPI;
221         locColMPI.resize(par.myLen);
222
223
224 #pragma omp parallel for default(none) shared(locColMPI, par) private(riC)
225         for (int locI = 0; locI < par.myLen; ++locI)
226         {
227             int i = locI + par.myDisp;
228
229             riC = 0.5 * (afl.r[i + 1] + afl.r[i]) - afl.rcm;
230
231             locColMPI[locI] = /*12 + 91*/ -0.5 * (riC ^ afl.tau[i]);
232
233             for (size_t j = 0; j < afl.np; ++j)
234             {
235                 locColMPI[locI] += /*92*/ IDPI * (Alpha(riC - afl.r[j + 1], riC -
    afl.r[j]) * afl.tau[j]
236                     - Lambda(riC - afl.r[j + 1], riC - afl.r[j]) *
    afl.nrm[j]
237                     )* afl.tau[i] * (riC ^ afl.tau[j]);
238
239                 locColMPI[locI] += /*93*/ IDPI * (Alpha(riC - afl.r[j + 1], riC -
    afl.r[j]) * afl.nrm[j] +
240                     Lambda(riC - afl.r[j + 1], riC - afl.r[j]) *
    afl.tau[j]
241                     )* afl.tau[i] * (riC ^ afl.nrm[j]);
242             }
243         }//pragma omp for
244         MPI_Gatherv(locColMPI.data(), par.myLen, MPI_DOUBLE, colMPI.data(), par.
    len.data(), par.disp.data(), MPI_DOUBLE, 0, W.getParallel().
    commWork);
245
246         if (id == 0)
247         for (size_t i = 0; i < (size_t)(col.rows()); ++i)
248         {
249             col(i, 0) = colMPI[i];
250         }
251     }//if (W.getCurrentStep() == 0)
252 }//FillAtt(...)
```

Граф вызовов:



Граф вызова функции:



10.14.3.3 FillMechanicsRhs()

void MechanicsRigidRotateMon::FillMechanicsRhs (
std::vector< double > & rhs) [virtual]

Заполнение правых частей, соответствующих механической системе

Аргументы

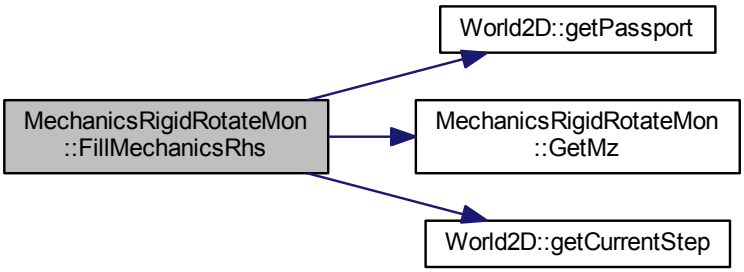
in	rhs	вектор правой части для механики одного профиля (количество элементов = количеству степеней свободы)

Замещает Mechanics.

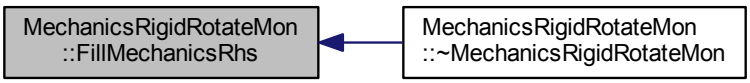
См. определение в файле MechanicsRigidRotateMon.cpp строка 175

```
{
    rhs[0] = /*7*/-J * Wcm + /*8*/ W.getPassport().timeDiscretizationProperties.dt * k * Phi -\
                /*11*/GetMz(W.getCurrentStep() * W.getPassport().timeDiscretizationProperties.dt)* W.getPassport().timeDiscretizationProperties.dt;

    Point2D riC;

#pragma warning (push)
#pragma warning (disable: 4101)
    double rhsOmp;
#pragma warning (pop)

#pragma omp parallel for default(none) shared(rhs) private(riC, rhsOmp)
    for (int loc_i = 0; loc_i < afl.np; ++loc_i)
    {
        riC = 0.5 * (afl.r[loc_i + 1] + afl.r[loc_i]) - afl.rcm;

        rhsOmp = 0.0;
        rhsOmp += /*5*/ 0.5 * riC * riC * afl.gammaThrough[loc_i];
        rhsOmp += /*6*/ 0.5 * riC * riC * Wcm * (riC ^ afl.tau[loc_i]) * afl.len[loc_i];
        rhsOmp += /*9*/ 0.5 * afl.len[loc_i] * (Wcm * riC ^ afl.tau[loc_i]) * (-Wcm * riC) ^ riC * W.getPassport().timeDiscretizationProperties.dt;
        rhsOmp -= /*10*/ -0.5 * Wcm * riC * riC * afl.len[loc_i] * (riC ^ afl.nrm[loc_i]) * Wcm * W.getPassport().timeDiscretizationProperties.dt;

#pragma omp atomic
        rhs[0] += rhsOmp;
    }
}//FillMechanicsRhs(...)
```

Граф вызовов:



Граф вызова функции:



10.14.3.4 FillMechanicsRowsAndCross()

```
void MechanicsRigidRotateMon::FillMechanicsRowsAndCross (
            Eigen::MatrixXd & row,
            Eigen::MatrixXd & cross )   [virtual]
```

Заполнение строк в матрице, соответствующих механической системе. А также пересечений строк и столбцов, соответствующих мех. системе

Аргументы

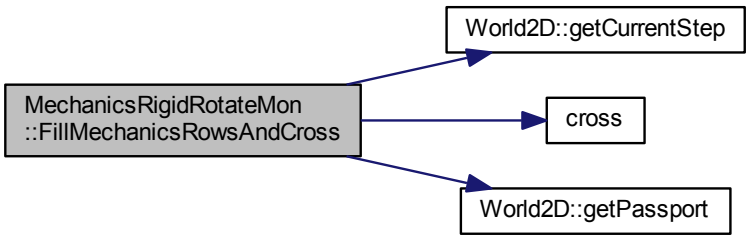
in	row	строки (количество строк = количество степеней свободы, количество элементов в строках = количество неизвестных в локальной матрице)
in	cross	пересечения (количество столбцов = количество степеней свободы, количество элементов в столбцах = количество степеней свободы)

Замещает Mechanics.

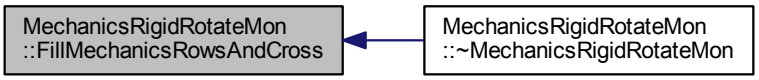
См. определение в файле MechanicsRigidRotateMon.cpp строка 148

```
149 {
150     if (W.getCurrentStep() == 0)
151     {
152         cross(0, 0) = -(/*3*/ J + /*4*/ W.getPassport().
    timeDiscretizationProperties.dt * b);
153
154         Point2D riC;
155 #pragma warning (push)
156 #pragma warning (disable: 4101)
157         double crossOmp;
158 #pragma warning (pop)
159
160 #pragma omp parallel for default(none) shared(row, cross) private(riC, crossOmp)
161         for (int loc_i = 0; loc_i < afl.np; ++loc_i)
162         {
163             riC = 0.5 * (afl.r[loc_i + 1] + afl.r[loc_i]) - afl.rcm;
164             crossOmp = /*2*/ 0.5 * riC * riC * (riC ^ afl.tau[loc_i]) *
    afl.len[loc_i];
165
166             row(0, loc_i) = /*1*/ 0.5 * riC * riC * afl.len[loc_i];
167 #pragma omp atomic
168             cross(0, 0) += crossOmp;
169         }//pragma omp for
170     }//if (W.getCurrentStep() == 0)
171 }//FillMechanicsRowsAndCols(...)
```

Граф вызовов:



Граф вызова функции:



10.14.3.5 GenerateForcesHeader()

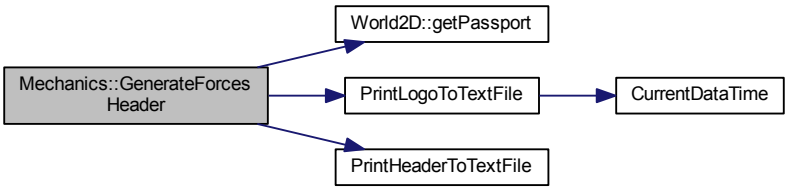
void Mechanics::GenerateForcesHeader ( ) [inherited]

Генерация заголовка файла нагрузок

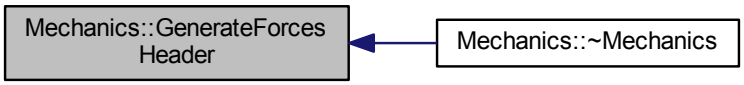
См. определение в файле Mechanics.cpp строка 69

```
{
  std::stringstream forceFileName, forceFileNameCsv;
  forceFileName << W.getPassport().dir << "forces-airfoil-" << numberInPassport;
  forceFileNameCsv << W.getPassport().dir << "forces-airfoil-" << numberInPassport << ".csv";

  std::ofstream newForcesFile(forceFileName.str());
  std::ofstream newForcesFileCsv(forceFileNameCsv.str());

  PrintLogoToTextFile(newForcesFile, forceFileName.str(), "Hydrodynamic loads for the airfoil " + W.getPassport().airfoilParams[numberInPassport].fileAirfoil);

  PrintHeaderToTextFile(newForcesFile, "currentStep     currentTime     Fx     Fy     Mz");

  newForcesFileCsv << "t,Fx,Fy,Mz";

  newForcesFile.close();
  newForcesFile.clear();

  newForcesFileCsv.close();
  newForcesFileCsv.clear();

}//GenerateForcesHeader()
```

Граф вызовов:



Граф вызова функции:



10.14.3.6 GenerateForcesString()

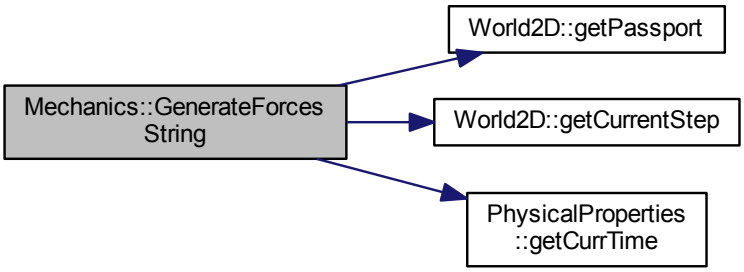
void Mechanics::GenerateForcesString ( ) [inherited]

Сохранение строки со статистикой в файл нагрузок

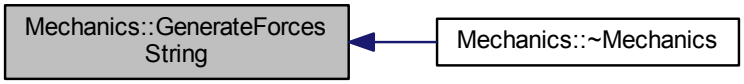
См. определение в файле Mechanics.cpp строка 121

```
122 {
123     std::stringstream forceFileName, forceFileNameCsv;
124     forceFileName << W.getPassport().dir << "forces-airfoil-" <<
    numberInPassport;
125     forceFileNameCsv << W.getPassport().dir << "forces-airfoil-" <<
    numberInPassport << ".csv";
126
127     std::ofstream forcesFile(forceFileName.str(), std::ios::app);
128     forcesFile << std::endl << W.getCurrentStep() << "  " << W.
    getPassport().physicalProperties.getCurrTime() << " " <<
    hydroDynamForce[0] << "  " << hydroDynamForce[1] << " " <<
    hydroDynamMoment;
129     forcesFile.close();
130
131     std::ofstream forcesFileCsv(forceFileNameCsv.str(), std::ios::app);
132     forcesFileCsv << std::endl << W.getPassport().physicalProperties.
    getCurrTime() << "," << hydroDynamForce[0] << "," <<
    hydroDynamForce[1] << "," << hydroDynamMoment;
133     forcesFileCsv.close();
134 }//GenerateForcesString()
```

Граф вызовов:



Граф вызова функции:



10.14.3.7 GeneratePositionHeader()

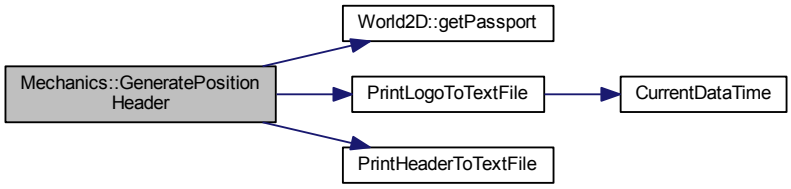
void Mechanics::GeneratePositionHeader ( ) [inherited]

Генерация заголовка файла положения профиля

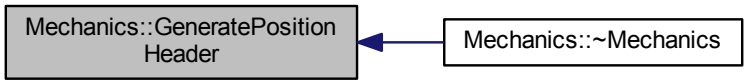
См. определение в файле Mechanics.cpp строка 94

```
95 {
96     if (isMoves)
97     {
98         std::stringstream positionFileName, positionFileNameCsv;
99         positionFileName << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport;
100         positionFileNameCsv << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport << ".csv";
101
102         std::ofstream newPositionFile(positionFileName.str());
103         std::ofstream newPositionFileCsv(positionFileNameCsv.str());
104
105         PrintLogoToTextFile(newPositionFile, positionFileName.str(), "Position of the
     airfoil " + W.getPassport().airfoilParams[numberInPassport].
    fileAirfoil);
106
107         PrintHeaderToTextFile(newPositionFile, "currentStep     currentTime     x
     y     phi     Vx     Vy     w");
108
109         newPositionFileCsv << "t,x,y,phi,Vx,Vy,w";
110
111         newPositionFile.close();
112         newPositionFile.clear();
113
114         newPositionFileCsv.close();
115         newPositionFileCsv.clear();
116     }
117 }//GeneratePositionHeader()
```

Граф вызовов:



Граф вызова функции:



10.14.3.8 GeneratePositionString()

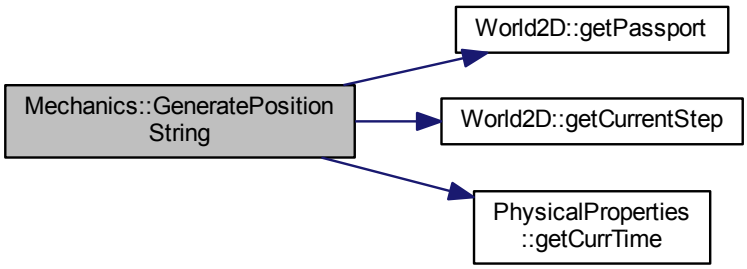
void Mechanics::GeneratePositionString ( ) [inherited]

Сохранение строки со статистикой в файл нагрузок

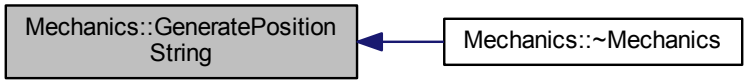
См. определение в файле Mechanics.cpp строка 138

```
139 {
140     if (isMoves)
141     {
142         std::stringstream positionFileName, positionFileNameCsv;
143         positionFileName << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport;
144         positionFileNameCsv << W.getPassport().dir << "position-airfoil-" <<
    numberInPassport << ".csv";
145
146         std::ofstream forcesFile(positionFileName.str(), std::ios::app);
147         forcesFile << std::endl << W.getCurrentStep() << "   " <<
    W.getPassport().physicalProperties.getCurrTime() << "  " <<
    afl.rcm[0] << "   " << afl.rcm[1] << "  " << Phi << "    " << Vcm[0] << " " <<
    Vcm[1] << "  " << Wcm;
148         forcesFile.close();
149
150         std::ofstream forcesFileCsv(positionFileNameCsv.str(), std::ios::app);
151         forcesFileCsv << std::endl << W.getPassport().
    physicalProperties.getCurrTime() << "," << afl.
    rcm[0] << "," << afl.rcm[1] << "," << Phi << "," << Vcm[0] << "," <<
    Vcm[1] << "," << Wcm;
152         forcesFileCsv.close();
153     }
154 }
```

Граф вызовов:



Граф вызова функции:



10.14.3.9 GetHydroDynamForce()

void MechanicsRigidRotateMon::GetHydroDynamForce (
timePeriod & time) [virtual]

Вычисление гидродинамической силы, действующей на профиль

Аргументы

out	time	ссылка на промежуток времени — пару чисел (время начала и время конца операции)

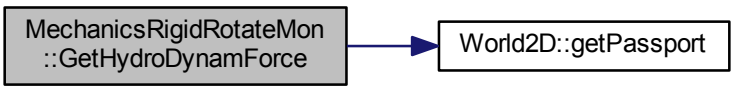
Замещает Mechanics.

См. определение в файле MechanicsRigidRotateMon.cpp строка 45

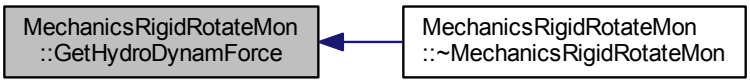
```
46 {
47     time.first = omp_get_wtime();
48
49     const double& dt = W.getPassport().timeDiscretizationProperties
   .dt;
50
51     hydroDynamForce = { 0.0, 0.0 };
52     hydroDynamMoment = 0.0;
53
54     Point2D hDFGam = { 0.0, 0.0 };       //гидродинамические силы, обусловленные Gamma_k
55     Point2D hDFdelta = { 0.0, 0.0 }; //гидродинамические силы, обусловленные delta_k
56     Point2D hDFQ = { 0.0, 0.0 };     //гидродинамические силы, обусловленные Q_k
57     double hDMGam = 0.0;               //гидродинамический момент, обусловленный Gamma_k
58     double hDMdelta = 0.0;             //гидродинамический момент, обусловленный Gamma_k
59     double hDMQ = 0.0;                 //гидродинамический момент, обусловленный Gamma_k
60
61     for (size_t i = 0; i < afl.np; ++i)
62     {
63         Point2D rK = 0.5 * (afl.r[i + 1] + afl.r[i]) - afl.rcm;
64         Point2D velK = { -Wcm * rK[1], Wcm * rK[0] };
65
66         double gAtt = Wcm * (rK ^ afl.tau[i]);
67         double gAttOld = WcmOld * (rK ^ afl.tau[i]);
68         double deltaGAtt = gAtt - gAttOld;
69
70         double qAtt = Wcm * (rK ^ afl.nrm[i]);
71
72         /*1*/
73         double deltaK = boundary.sheets.freeVortexSheet[i][0] *
   afl.len[i] - afl.gammaThrough[i] + deltaGAtt * afl.len[i];
74         hDFdelta += deltaK * Point2D({ -rK[1], rK[0] });
75         hDMdelta += 0.5 * deltaK * (rK * rK);
76
77         /*2*/
78         hDFGam += 0.5 * afl.v[i].kcross() * gAtt * afl.len[i];
79         hDMGam += 0.5 * rK ^ afl.v[i].kcross() * gAtt * afl.len[i];
```

```
80
81          /*3*/
82          hDFQ -= 0.5 * afl.v[i] * qAtt * afl.len[i];
83          hDMQ -= 0.5 * rK ^ afl.v[i] * qAtt * afl.len[i];
84      }
85
86      hydroDynamForce = hDFGam + hDFdelta * (1.0 / dt) + hDFQ;
87      hydroDynamMoment = hDMGam + hDMdelta / dt + hDMQ;
88
89      time.second = omp_get_wtime();
90 }// CalcHydroDynamForce(...)
```

Граф вызовов:



Граф вызова функции:



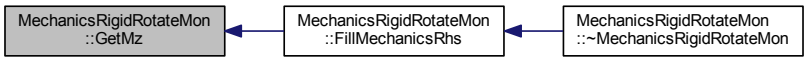
10.14.3.10 GetMz()

double MechanicsRigidRotateMon::GetMz (
double t) [inline], [private]

См. определение в файле MechanicsRigidRotateMon.h строка 97

```
98      {
99          if (t > (tMomentAccel + tRotateAccel))
100             return Mz;
101         else if (t < tRotateAccel)
102             return 0.0;
103         else return ((Mz / tMomentAccel) * t - (Mz *
    tRotateAccel / tMomentAccel));
104     }
```

Граф вызова функции:



10.14.3.11 Lambda()

double MechanicsRigidRotateMon::Lambda (
const Point2D & p,
const Point2D & s) const [inline], [private]

Вспомогательная функция вычисления логарифма отношения норм векторов

Аргументы

in	p	константная ссылка на первый вектор
in	s	константная ссылка на второй вектор

Возвращает

логарифм отношения норм векторов

См. определение в файле MechanicsRigidRotateMon.h строка 92

```
93    {
94        return 0.5*log((s*s) / (p*p));
95    }
```

Граф вызова функции:

10.14.3.12 Move()

void MechanicsRigidRotateMon::Move ( ) [virtual]

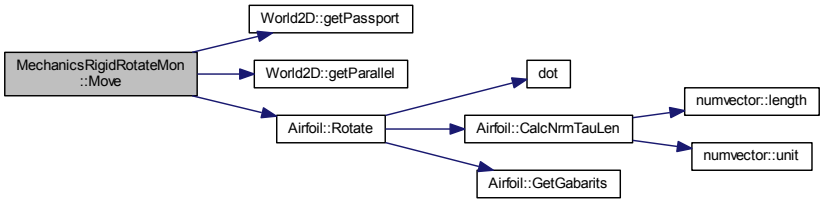
Перемещение профиля в соответствии с законом

Замещает Mechanics.

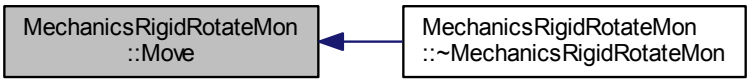
См. определение в файле MechanicsRigidRotateMon.cpp строка 116

```
117 {
118     PhiOld = Phi;
119
120     double dphi = Wcm * W.getPassport().
    timeDiscretizationProperties.dt;
121     MPI_Bcast(&dphi, 1, MPI_DOUBLE, 0, W.getParallel().commWork);
122
123     Phi += dphi;
124     afl.Rotate(dphi);
125 }//Move()
```

Граф вызовов:



Граф вызова функции:



10.14.3.13 PositionOfAirfoilRcm()

Point2D MechanicsRigidRotateMon::PositionOfAirfoilRcm (
double currTime) [virtual]

Вычисление положения центра масс профиля

Аргументы

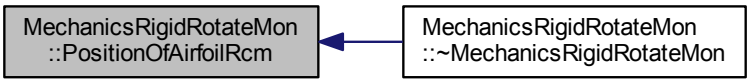
in	currTime	текущее время

Замещает Mechanics.

См. определение в файле MechanicsRigidRotateMon.cpp строка 100

```
101 {
102     return afl.rcm;
103 }//PositionOfAirfoilRcm(...)
```

Граф вызова функции:



10.14.3.14 ReadSpecificParametersFromDictionary()

void MechanicsRigidRotateMon::ReadSpecificParametersFromDictionary ( ) [virtual]

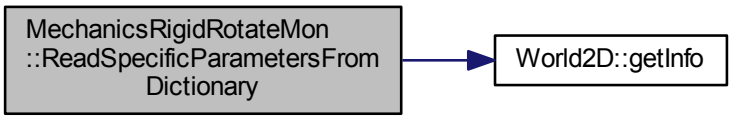
Чтение параметров конкретной механической системы

Замещает Mechanics.

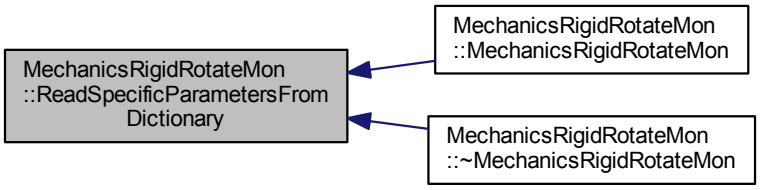
См. определение в файле MechanicsRigidRotateMon.cpp строка 128

```
129 {
130     mechParamsParser->get("J", J);
131     mechParamsParser->get("k", k);
132     mechParamsParser->get("tRotateAccel", tRotateAccel);
133     mechParamsParser->get("tMomentAccel", tMomentAccel);
134     mechParamsParser->get("Mz", Mz);
135
136     W.getInfo('i') << "J = " << J << std::endl;
137     W.getInfo('i') << "k = " << k << std::endl;
138     W.getInfo('i') << "tRotateAccel = " << tRotateAccel << std::endl;
139     W.getInfo('i') << "tMomentAccel = " << tMomentAccel << std::endl;
140     W.getInfo('i') << "Mz = " << Mz << std::endl;
141
142 //  std::vector<double> sh;
143 //  mechParamsParser->get("sh", sh);
144 //  k = 0.0 * m * 4.0 * PI * PI * sh[1] * sh[1] * W.getPassport().physicalProperties.V0().length2();
145 }
```

Граф вызовов:



Граф вызова функции:



10.14.3.15 SolutionToMechanicalSystem()

void MechanicsRigidRotateMon::SolutionToMechanicalSystem (
Eigen::VectorXd & col) [virtual]

Извлечение из решения параметров механической системы

Аргументы

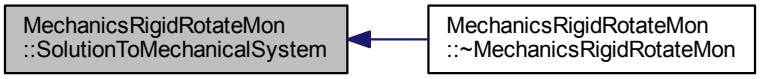
in	col	

Замещает Mechanics.

См. определение в файле MechanicsRigidRotateMon.cpp строка 255

```
{
    WcmOld = Wcm;

    Wcm = col(0);

    for (size_t i = 0; i < afl.np + 1; ++i)
        afl.v[i] = afl.r[i].kcross() * Wcm;
}
```

Граф вызова функции:



10.14.3.16 VeloOfAirfoilPanels()

void MechanicsRigidRotateMon::VeloOfAirfoilPanels (
double currTime) [virtual]

Вычисление скоростей начал панелей

Аргументы

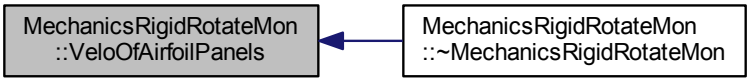
in	currTime	текущее время

Замещает Mechanics.

См. определение в файле MechanicsRigidRotateMon.cpp строка 106

```
107 {
108     for (size_t i = 0; i < afl.v.size(); ++i)
109     {
110         afl.v[i][0] = afl.r[i].kcross()[0] * Wcm;
111         afl.v[i][1] = afl.r[i].kcross()[1] * Wcm;
112     }
113 }//VeloOfAirfoilPanels(...)
```

Граф вызова функции:



10.14.3.17 VeloOfAirfoilRcm()

Point2D MechanicsRigidRotateMon::VeloOfAirfoilRcm (
double currTime) [virtual]

Вычисление скорости центра масс профиля

Аргументы

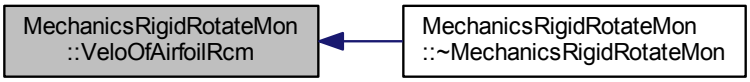
in	currTime	текущее время

Замещает Mechanics.

См. определение в файле MechanicsRigidRotateMon.cpp строка 93

```
94 {
95     return{ 0.0, 0.0 };
96     //return{ 0.0, 0.0 };
97 }//VeloOfAirfoilRcm(...)
```

Граф вызова функции:



10.14.4 Данные класса

10.14.4.1 afl

Airfoil& Mechanics::afl [protected], [inherited]

Константная ссылка на профиль
инициализируется автоматом в конструкторе, при помощи const_cast

Предупреждения

использует const_cast для получения неконстантной ссылки

См. определение в файле Mechanics.h строка 83

10.14.4.2 b

double MechanicsRigidRotateMon::b [private]

См. определение в файле MechanicsRigidRotateMon.h строка 65

10.14.4.3 boundary

const Boundary& Mechanics::boundary [protected], [inherited]

Константная ссылка на граничное условие
инициализируется автоматом в конструкторе

См. определение в файле Mechanics.h строка 87

10.14.4.4 degOfFreedom

const size_t Mechanics::degOfFreedom [inherited]

Количество степеней свободы

См. определение в файле Mechanics.h строка 125

10.14.4.5 hydroDynamForce

Point2D Mechanics::hydroDynamForce [inherited]

Вектор гидродинамической силы и момент, действующие на профиль

См. определение в файле Mechanics.h строка 128

10.14.4.6 hydroDynamMoment

double Mechanics::hydroDynamMoment [inherited]

См. определение в файле Mechanics.h строка 129

10.14.4.7 isDeform

const bool Mechanics::isDeform [inherited]

Переменная, отвечающая за то, деформируется профиль или нет

См. определение в файле Mechanics.h строка 119

10.14.4.8 isMoves

const bool Mechanics::isMoves [inherited]

Переменная, отвечающая за то, двигается профиль или нет

См. определение в файле Mechanics.h строка 116

10.14.4.9 isRotate

const bool Mechanics::isRotate [inherited]

Переменная, отвечающая за то, может профиль вращаться или нет

См. определение в файле Mechanics.h строка 122

10.14.4.10 J

double MechanicsRigidRotateMon::J [private]

См. определение в файле MechanicsRigidRotateMon.h строка 64

10.14.4.11 k

double MechanicsRigidRotateMon::k [private]

См. определение в файле MechanicsRigidRotateMon.h строка 66

10.14.4.12 mechParamsParser

std::unique_ptr<StreamParser> Mechanics::mechParamsParser [protected], [inherited]

Умный указатель на парсер параметров механической системы

См. определение в файле Mechanics.h строка 94

10.14.4.13 Mz

double MechanicsRigidRotateMon::Mz [private]

Момент, который снимается с ротора

См. определение в файле MechanicsRigidRotateMon.h строка 75

10.14.4.14 numberInPassport

const size_t Mechanics::numberInPassport [protected], [inherited]

Номер профиля в паспорте

См. определение в файле Mechanics.h строка 78

10.14.4.15 Phi

double Mechanics::Phi [protected], [inherited]

См. определение в файле Mechanics.h строка 106

10.14.4.16 Phi0

double Mechanics::Phi0 [protected], [inherited]

См. определение в файле Mechanics.h строка 100

10.14.4.17 PhiOld

double Mechanics::PhiOld [protected], [inherited]

См. определение в файле Mechanics.h строка 112

10.14.4.18 Rcm

Point2D Mechanics::Rcm [protected], [inherited]

Текущие положение профиля

См. определение в файле Mechanics.h строка 106

10.14.4.19 Rcm0

const Point2D Mechanics::Rcm0 [protected], [inherited]

Начальное положение профиля

См. определение в файле Mechanics.h строка 100

10.14.4.20 RcmOld

Point2D Mechanics::RcmOld [protected], [inherited]

Текущие положение профиля

См. определение в файле Mechanics.h строка 112

10.14.4.21 tMomentAccel

double MechanicsRigidRotateMon::tMomentAccel [private]

время, в течение которого постепенно увеличивается момент, которым нагружается ротор

См. определение в файле MechanicsRigidRotateMon.h строка 72

10.14.4.22 tRotateAccel

double MechanicsRigidRotateMon::tRotateAccel [private]

время, в течение которого ротор крутится без нагрузки

См. определение в файле MechanicsRigidRotateMon.h строка 69

10.14.4.23 Vcm

Point2D Mechanics::Vcm [protected], [inherited]

Текущие скорость центра и угловая скорость

См. определение в файле Mechanics.h строка 103

10.14.4.24 Vcm0

const Point2D Mechanics::Vcm0 [protected], [inherited]

Начальная скорость центра и угловая скорость

См. определение в файле Mechanics.h строка 97

10.14.4.25 VcmOld

Point2D Mechanics::VcmOld [protected], [inherited]

Скорость и отклонение с предыдущего шага

См. определение в файле Mechanics.h строка 109

10.14.4.26 virtVortParams

const VortexesParams& Mechanics::virtVortParams [protected], [inherited]

Константная ссылка на структуру с параметрами виртуального вихревого слоя для профиля инициализируется автоматом в конструкторе

См. определение в файле Mechanics.h строка 91

10.14.4.27 W

const World2D& Mechanics::W [protected], [inherited]

Константная ссылка на решаемую задачу

См. определение в файле Mechanics.h строка 75

10.14.4.28 Wcm

double Mechanics::Wcm [protected], [inherited]

См. определение в файле Mechanics.h строка 103

10.14.4.29 Wcm0

const double Mechanics::Wcm0 [protected], [inherited]

См. определение в файле Mechanics.h строка 97

10.14.4.30 WcmOld

double Mechanics::WcmOld [protected], [inherited]

См. определение в файле Mechanics.h строка 109

Объявления и описания членов классов находятся в файлах:

- Mechanics/MechanicsRigidRotateMon.h
- Mechanics/MechanicsRigidRotateMon.cpp

10.15 Структура NumericalSchemes

Структура, задающая используемые численные схемы

#include <Passport.h>

Открытые атрибуты

- int linearSystemSolver
- int velocityComputation
- int wakeMotionIntegrator

10.15.1 Подробное описание

Структура, задающая используемые численные схемы

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Passport.h строка 209

10.15.2 Данные класса

10.15.2.1 linearSystemSolver

int NumericalSchemes::linearSystemSolver

См. определение в файле Passport.h строка 212

10.15.2.2 velocityComputation

int NumericalSchemes::velocityComputation

См. определение в файле Passport.h строка 215

10.15.2.3 wakeMotionIntegrator

int NumericalSchemes::wakeMotionIntegrator

См. определение в файле Passport.h строка 218

Объявления и описания членов структуры находятся в файле:

- Passport/Passport.h

10.16 Шаблон класса nummatrix< T, n, m >

Шаблонный класс, определяющий матрицу фиксированного размера
Фактически представляет собой массив, для которого определено большое количество различных операций.
Для доступа к элементам матрицы используется оператор [][]
Оператор[] имитирует возврат ссылки/константной ссылки на numvector.

#include <nummatrix.h>

Открытые члены

- numvector< T, m > & operator[] (size_t i)

  Перегрузка оператора "[]" доступа к строке
- const numvector< T, m > & operator[] (size_t i) const

  Перегрузка оператора "[]" доступа к строке
- template<typename P >
  nummatrix< T, n, m > & operator*= (const P c)

  Перегрузка оператора "*=" домножения матрицы на действительное число
- template<typename P >
  nummatrix< T, n, m > & operator/= (const P c)

  Перегрузка оператора "/=" деления матрицы на действительное число
- template<typename P >
  nummatrix< T, n, m > & operator+= (const nummatrix< P, n, m > &y)

  Перегрузка оператора "+=" прибавления другой матрицы
- template<typename P >
  nummatrix< T, n, m > & operator-= (const nummatrix< P, n, m > &y)

  Перегрузка оператора "-=" вычитания другой матрицы
- template<typename P >
  auto operator+ (const nummatrix< P, n, m > &y) const -> nummatrix< std::remove_const< decltype(r[0]+y.r[0])>::type, n, m >

  Перегрузка оператора "+" сложения двух матриц
- template<typename P >
  auto operator- (const nummatrix< P, n, m > &y) const -> nummatrix< std::remove_const< decltype(r[0] - y.r[0])>::type, n, m >

  Перегрузка оператора "-" вычитания двух матриц
- template<typename P >
  auto operator* (const P c) const -> nummatrix< std::remove_const< decltype(r[0] *c)>::type, n, m >

  Перегрузка оператора "*" умножения матрицы справа на число
- nummatrix< T, n, m > operator- () const

  Перегрузка оператора "-" унарного минуса
- template<typename P >
  bool operator== (const nummatrix< P, n, m > &y) const

  Перегрузка оператора "==" логического равенства
- template<typename P >
  bool operator!= (const nummatrix< P, n, m > &y) const

  Перегрузка оператора "!=" логического неравенства
- std::pair< size_t, size_t > size () const

  Вычисление размерности матрицы (числа строк и столбцов в ней)
- auto norm1 () const -> std::remove_const< std::remove_reference< decltype(r[0])>::type >::type

  Вычисление 1-нормы матрицы
- auto norminf () const -> std::remove_const< std::remove_reference< decltype(r[0])>::type >::type

  Вычисление inf-нормы матрицы
- nummatrix< T, n, m > & toZero (T val=0.0)
- nummatrix< T, n, n > & toIdentity ()
- nummatrix ()

  Пустой конструктор
- template<typename P >
  nummatrix (const T z)

  Конструктор, инициализирующий всю матрицу одной и той же константой
- template<typename P >
  nummatrix (const nummatrix< P, n, m > &mtr)

Конструктор копирования

- template<typename P >
  nummatrix (const std::vector< std::vector< P >> &vec)

  Конструктор инициализации с помощью std::vector из std::vector.
- template<typename P >
  nummatrix (const numvector< numvector< P, m >, n > &mtr)

  Конструктор инициализации при помощи numvector из numvector.
- nummatrix (const std::initializer_list< numvector< T, m >> &z)

  Нешаблонный конструктор инициализации списком из numvector.
- template<typename P >
  nummatrix (const std::initializer_list< numvector< P, m >> &z)

  Шаблонный конструктор инициализации списком из numvector.
- operator std::vector< std::vector< T >> () const

  Приведение матрицы к типу std::vector из std::vector.
- nummatrix< T, m, n > transpose ()

  Транспонирование
- template<typename P >
  auto operator & (const numvector< P, m > &x) -> numvector< std::remove_const< decltype(r[0] ∗x[0])>::type, n >

  Умножение матрицы на вектор
- template<typename P , size_t p>
  auto operator & (const nummatrix< P, m, p > &B) -> nummatrix< std::remove_const< decltype(r[0] ∗x[0])>::type, n, p >

  Умножение матрицы на матрицу

Открытые атрибуты

- T r [n ∗m]

  Собственно, вектор с данными

10.16.1 Подробное описание

template<typename T, size_t n, size_t m>
class nummatrix< T, n, m >

Шаблонный класс, определяющий матрицу фиксированного размера
Фактически представляет собой массив, для которого определено большое количество различных операций.
Для доступа к элементам матрицы используется оператор [][]
Оператор[] имитирует возврат ссылки/константной ссылки на numvector.

Template Parameters

T	тип элементов матрицы
n	число строк
m	число столбцов

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле nummatrix.h строка 68

10.16.2 Конструктор(ы)

10.16.2.1 nummatrix() [1/7]

template<typename T, size_t n, size_t m>
nummatrix< T, n, m >::nummatrix ( ) [inline]

Пустой конструктор

См. определение в файле nummatrix.h строка 343

```
343 { };
```

Граф вызова функции:

10.16.2.2 nummatrix() [2/7]

template<typename T, size_t n, size_t m>
template<typename P >
nummatrix< T, n, m >::nummatrix (
const T z) [inline], [explicit]

Конструктор, инициализирующий всю матрицу одной и той же константой

Template Parameters

P	тип данных инициализирующей константы

Аргументы

in	z	значение, которым инициализируются все компоненты матрицы

См. определение в файле nummatrix.h строка 351

```
352     {
353         for (size_t i = 0; i < n*m; ++i)
354             r[i] = z;
355     }//nummatrix(...)
```

10.16.2.3 nummatrix() [3/7]

```
template<typename T, size_t n, size_t m>
template<typename P >
nummatrix< T, n, m >::nummatrix (
                const nummatrix< P, n, m > & mtr )   [inline]
```

Конструктор копирования

Template Parameters

P	тип данных копируемой матрицы
n	число строк
m	число столбцов

Аргументы

in	mtr	константная ссылка на копируемую матрицу

См. определение в файле nummatrix.h строка 365

```
366     {
367         for (size_t i = 0; i < n*m; ++i)
368             r[i] = mtr.r[i];
369     }//nummatrix(...)
```

10.16.2.4 nummatrix() [4/7]

```
template<typename T, size_t n, size_t m>
template<typename P >
```

nummatrix< T, n, m >::nummatrix (
const std::vector< std::vector< P >> & vec) [inline]

Конструктор инициализации с помощью std::vector из std::vector.

Template Parameters

P	тип данных инициализирующего std::vector
n	число строк
m	число столбцов

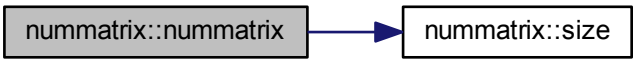
Аргументы

in	vec	константная ссылка на инициализирующий вектор

См. определение в файле nummatrix.h строка 379

```
380    {
381        if ((vec.size() != n) || (vec[0].size() != m))
382            throw;
383        for (size_t i = 0; i < n; ++i)
384        for (size_t j = 0; j < m; ++j)
385            r[i*m + j] = vec[i][j];
386    }//nummatrix(...)
```

Граф вызовов:



10.16.2.5 nummatrix() [5/7]

```
template<typename T, size_t n, size_t m>
template<typename P >
nummatrix< T, n, m >::nummatrix (
                const numvector< numvector< P, m >, n > & mtr )   [inline]
```

Конструктор инициализации при помощи numvector из numvector.

Template Parameters

P	тип данных в инициализирующем numvector
n	число строк
m	число столбцов

Аргументы

in	mtr	константная ссылка на numvector из numvector

См. определение в файле nummatrix.h строка 396

```
397     {
398         for (size_t i = 0; i < n; ++i)
399         for (size_t j = 0; j < m; ++j)
400             r[i*m + j] = mtr[i][j];
401     }//nummatrix(...)
```

Граф вызовов:

10.16.2.6 nummatrix() [6/7]

```
template<typename T, size_t n, size_t m>
nummatrix< T, n, m >::nummatrix (
            const std::initializer_list< numvector< T, m >> & z )   [inline]
```

Нешаблонный конструктор инициализации списком из numvector.

Template Parameters

T	тип данных

Аргументы

in	z	константная ссылка на список инициализации

Предупреждения

Длина списка инициализации проверяется, при несовпадении бросается исключение

См. определение в файле nummatrix.h строка 410

```
411     {
412         if (z.size() != n)
```

```
            throw;
        for (size_t i = 0; i < n; ++i)
        {
            const numvector<T, m>& numv = *(z.begin() + i);
            for (size_t j = 0; j < m; ++j)
                r[i*m + j] = numv[j];
        }
    }//nummatrix(...)
```

10.16.2.7 nummatrix() [7/7]

template<typename T, size_t n, size_t m>
template<typename P >
nummatrix< T, n, m >::nummatrix (
const std::initializer_list< numvector< P, m >> & z) [inline]

Шаблонный конструктор инициализации списком из numvector.

Template Parameters

P	тип данных векторов в списке инициализации

Аргументы

in	z	константная ссылка на список инициализации

Предупреждения

Длина списка инициализации проверяется, при несовпадении бросается исключение

См. определение в файле nummatrix.h строка 429

```
    {
        if (z.size() != n)
            throw;
        for (size_t i = 0; i < n; ++i)
        {
            const numvector<T, m>& numv = *(z.begin() + i);
            for (size_t j = 0; j < m; ++j)
                r[i*m + j] = numv[j];
        }
    }//nummatrix(...)
```

10.16.3 Методы

10.16.3.1 norm1()

template<typename T, size_t n, size_t m>
auto nummatrix< T, n, m >::norm1 ( ) const -> std::remove_const<std::remove_reference<decltype(r[0])>::type>::type [inline]

Вычисление 1-нормы матрицы

Возвращает

1-норма матрицы

См. определение в файле nummatrix.h строка 291

```
{
    std::vector<std::remove_const<std::remove_reference<decltype(r[0])>::type>::type> ms(m, 0);
    for (size_t i = 0; i < m; ++i)
    for (size_t j = 0; j < n; ++j)
        ms[i] += abs(r[j*m + i]);
    return std::max_element(ms.begin(), ms.end());
}
```

10.16.3.2 norminf()

template<typename T, size_t n, size_t m>
auto nummatrix< T, n, m >::norminf ( ) const -> std::remove_const<std::remove_reference<decltype(r[0])>::type>::type [inline]

Вычисление inf-нормы матрицы

Возвращает

inf-норма матрицы

См. определение в файле nummatrix.h строка 303

```
{
    std::vector<std::remove_const<std::remove_reference<decltype(r[0])>::type>::type> ms(n, 0);
    for (size_t i = 0; i < n; ++i)
    for (size_t j = 0; j < m; ++j)
        ms[i] += abs(r[i*m + j]);
    return std::max_element(ms.begin(), ms.end());
}
```

10.16.3.3 operator &() [1/2]

template<typename T, size_t n, size_t m>
template<typename P >
auto nummatrix< T, n, m >::operator& (
const numvector< P, m > & x) -> numvector<std::remove_const<decltype(r[0] * x[0])>::type, n> [inline]

Умножение матрицы на вектор

Template Parameters

P	тип данных вектора
n	число строк матрицы
m	число столбцов матрицы и длина вектора

Аргументы

in	x	константная ссылка на вектор

Возвращает

вектор — результат умножения матрицы на вектор

См. определение в файле nummatrix.h строка 479

```
481     {
482         numvector<std::remove_const<decltype(r[0] * x[0])>::type
    , n> res;
483
484         for (size_t i = 0; i < n; ++i)
485             res[i] = (*this)[i] & x;
486
487         return res;
488     }//operator&
```

10.16.3.4 operator &() [2/2]

```
template<typename T, size_t n, size_t m>
template<typename P , size_t p>
auto nummatrix< T, n, m >::operator& (
                    const nummatrix< P, m, p > & B ) -> nummatrix<std::remove_const<decltype(r[0] * x[0])>::type, n, p>
[inline]
```

Умножение матрицы на матрицу

Template Parameters

P	тип данных матрицы - второго множителя
n	число строк матрицы
m	число столбцов матрицы и длина вектора

Аргументы

in	B	константная ссылка на матрицу

Возвращает

вектор — результат умножения матрицы на вектор

См. определение в файле nummatrix.h строка 498

```
500     {
501         nummatrix<std::remove_const<decltype(r[0] * x[0])>::type
    , n, p> res(0);
502
```

```
503         for (size_t i = 0; i < n; ++i)
504         for (size_t j = 0; j < n; ++j)
505         for (size_t k = 0; k < n; ++k)
506             res[i][j] += (*this)[i][k] * B[k][j];
507         return res;
508     }//operator&
```

10.16.3.5 operator std::vector< std::vector< T >>()

template<typename T, size_t n, size_t m>
nummatrix< T, n, m >::operator std::vector< std::vector< T >> ( ) const [inline]

Приведение матрицы к типу std::vector из std::vector.

Возвращает

вектор типа std::vector, состоящий из std::vector, являющихся строками матрицы

См. определение в файле nummatrix.h строка 446

```
447     {
448         std::vector<std::vector<T>> vec;
449         vec.reserve(n);
450         for (size_t i = 0; i < n; ++i)
451         {
452             vec[i].reserve(m);
453             for (size_t j = 0; j < m; ++j)
454                 vec[i].push_back(r[i*m + j]);
455         }
456         return vec;
457     }
```

10.16.3.6 operator"!=()

template<typename T, size_t n, size_t m>
template<typename P >
bool nummatrix< T, n, m >::operator!= (
const nummatrix< P, n, m > & y) const [inline]

Перегрузка оператора "!=" логического неравенства

Template Parameters

T	тип данных
P	тип данных матрицы, с которой производится сравнение
n	число строк
m	число столбцов

Аргументы

in	y	константная ссылка на сравниваемую матрицу

Возвращает

true, если матрицы различаются, false в противном случае

См. определение в файле nummatrix.h строка 276

```
{
    return !(*this == y);
}//operator!=
```

10.16.3.7 operator∗()

template<typename T, size_t n, size_t m>
template<typename P >
auto nummatrix< T, n, m >::operator∗ (
const P c) const -> nummatrix<std::remove_const<decltype(r[0] ∗ c)>::type, n, m> [inline]

Перегрузка оператора "∗" умножения матрицы справа на число

Template Parameters

T	тип данных
n	число строк
m	число столбцов

Аргументы

in	c	число-множитель

Возвращает

результат умножения вектора на число, приведенный к соответствующему типу

См. определение в файле nummatrix.h строка 225

```
{
    nummatrix<std::remove_const<decltype(r[0] * c)>::type
, n, m> res;
    for (size_t i = 0; i < n*m; ++i)
        res[0][i] = c * r[i];
    return res;
}//operator*
```

10.16.3.8 operator∗=()

template<typename T, size_t n, size_t m>
template<typename P >
nummatrix<T, n, m>& nummatrix< T, n, m >::operator∗= (
const P c) [inline]

Перегрузка оператора "∗=" домножения матрицы на действительное число

Template Parameters

T	тип данных
P	тип множителя
n	число строк
m	число столбцов

Аргументы

in	c	числовой множитель типа, приводимого к типу компонент матрицы

Возвращает

ссылка на саму себя после домножения на число

См. определение в файле nummatrix.h строка 122

```
123     {
124         for (size_t i = 0; i < n*m; ++i)
125             r[i] *= c;
126         return *this;
127     }//operator*=
```

10.16.3.9 operator+()

template<typename T, size_t n, size_t m>
template<typename P >
auto nummatrix< T, n, m >::operator+ (
const nummatrix< P, n, m > & y) const -> nummatrix<std::remove_const<decltype(r[0] + y.r[0])>::type, n, m> [inline]

Перегрузка оператора "+" сложения двух матриц

Template Parameters

T	тип данных
P	тип данных матрицы - второго слагаемого
n	число строк
m	число столбцов

Аргументы

in	y	константная ссылка на прибавляемую матрицу

Возвращает

результат сложения двух матриц, приведенный к нужному типу

См. определение в файле nummatrix.h строка 190

```
191 {
192     nummatrix<std::remove_const<decltype(r[0] + y.r[0])>::type, n, m> res;
193     for (size_t i = 0; i < n*m; ++i)
194         res[0][i] = r[i] + y[i];
195     return res;
196 }//operator+
```

10.16.3.10 operator+=()

```
template<typename T, size_t n, size_t m>
template<typename P >
nummatrix<T, n, m>& nummatrix< T, n, m >::operator+= (
                const nummatrix< P, n, m > & y )   [inline]
```

Перегрузка оператора "+=" прибавления другой матрицы

Template Parameters

T	тип данных
P	тип данных прибавляемой матрицы
n	число строк
m	число столбцов

Аргументы

in	y	константная ссылка на прибавляемую матрицу

Возвращает

ссылка на саму себя после сложения с другой матрицей

См. определение в файле nummatrix.h строка 156

```
157 {
158     for (size_t i = 0; i < n*m; ++i)
159         r[i] += y[i];
160     return *this;
161 }//operator+=
```

10.16.3.11 operator-() [1/2]

```
template<typename T, size_t n, size_t m>
template<typename P >
auto nummatrix< T, n, m >::operator- (
            const nummatrix< P, n, m > & y ) const -> nummatrix<std::remove_const<decltype(r[0] - y.r[0])>::type,
n, m>    [inline]
```

Перегрузка оператора "-" вычитания двух матриц

Template Parameters

T	тип данных
P	тип данных матрицы - вычитаемого
n	число строк
m	число столбцов

Аргументы

in	y	константная ссылка на вычитаемую матрицу

Возвращает

результат вычитания двух матриц, приведенный к нужному типу

См. определение в файле nummatrix.h строка 208

```
209    {
210        nummatrix<std::remove_const<decltype(r[0] - y.r[0])>::type, n, m> res;
211        for (size_t i = 0; i < n*m; ++i)
212            res[0][i] = r[i] - y[i];
213        return res;
214    }//operator-
```

10.16.3.12 operator-() [2/2]

```
template<typename T, size_t n, size_t m>
nummatrix<T, n, m> nummatrix< T, n, m >::operator- ( ) const    [inline]
```

Перегрузка оператора "-" унарного минуса

Template Parameters

T	тип данных
n	число строк
m	число столбцов

Возвращает

противоположная матрица

См. определение в файле nummatrix.h строка 240

```
{
    nummatrix<T, n, m> res;
    for (size_t i = 0; i < n*m; ++i)
        res[0][i] = -r[i];
    return res;
}//operator-
```

10.16.3.13 operator-=()

```
template<typename T, size_t n, size_t m>
template<typename P >
nummatrix<T, n, m>& nummatrix< T, n, m >::operator-= (
                const nummatrix< P, n, m > & y )   [inline]
```

Перегрузка оператора "-=" вычитания другой матрицы

Template Parameters

T	тип данных
P	тип данных вычитаемой матрицы
n	число строк
m	число столбцов

Аргументы

in	y	константная ссылка на вычитаемую матрицу

Возвращает

ссылка на саму себя после вычитания другой матрицы

См. определение в файле nummatrix.h строка 173

```
{
    for (size_t i = 0; i < n*m; ++i)
        r[i] -= y[i];
    return *this;
}//operator-=
```

10.16.3.14 operator/=()

```
template<typename T, size_t n, size_t m>
template<typename P >
nummatrix<T, n, m>& nummatrix< T, n, m >::operator/= (
                const P c )  [inline]
```

Перегрузка оператора "/=" деления матрицы на действительное число

Template Parameters

T	тип данных
P	тип множителя
n	число строк
m	число столбцов

Аргументы

in	c	числовой делитель типа, приводимого к типу компонент матрицы

Возвращает

ссылка на саму себя после деления на число

См. определение в файле nummatrix.h строка 139

```
140     {
141         for (size_t i = 0; i < n*m; ++i)
142             r[i] /= c;
143         return *this;
144     }//operator/=
```

10.16.3.15 operator==()

```
template<typename T, size_t n, size_t m>
template<typename P >
bool nummatrix< T, n, m >::operator== (
                    const nummatrix< P, n, m > & y ) const   [inline]
```

Перегрузка оператора "==" логического равенства

Template Parameters

T	тип данных
P	тип данных матрицы, с которой производится сравнение
n	число строк
m	число столбцов

Аргументы

in	y	константная ссылка на сравниваемую матрицу

Возвращает

true, если матрицы одинаковые, false в противном случае

См. определение в файле nummatrix.h строка 258

```
259     {
260         for (size_t i = 0; i < n*m; ++i)
261         if (r[i] != y.r[i])
262             return false;
263         return true;
264     }//operator==
```

10.16.3.16 operator[]() [1/2]

```
template<typename T, size_t n, size_t m>
numvector<T, m>& nummatrix< T, n, m >::operator[] (
                size_t i )   [inline]
```

Перегрузка оператора "[]" доступа к строке

Template Parameters

T	тип данных

Аргументы

in	i	номер строки, к которой происходит обращение

Возвращает

Ссылка на numvector, соответствующий i-й строке матрицы

См. определение в файле nummatrix.h строка 81

```
82     {
83         return *(reinterpret_cast<numvector<T, m>*>(r + (i*m)));
84     }
```

10.16.3.17 operator[]() [2/2]

```
template<typename T, size_t n, size_t m>
const numvector<T, m>& nummatrix< T, n, m >::operator[] (
                size_t i ) const   [inline]
```

Перегрузка оператора "[]" доступа к строке

Template Parameters

T	тип данных

Аргументы

in	i	номер строки, к которой происходит обращение

Возвращает

Константная сылка на numvector, соответствующий i-й строке матрицы

См. определение в файле nummatrix.h строка 91

```
92    {
93        return *(reinterpret_cast<const numvector<T, m>*>(r + (i*m)));
94    }
```

10.16.3.18 size()

template<typename T, size_t n, size_t m>
std::pair<size_t, size_t> nummatrix< T, n, m >::size ( ) const [inline]

Вычисление размерности матрицы (числа строк и столбцов в ней)

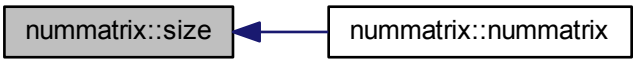
Возвращает

размерность вектора (число элементов в нем)

См. определение в файле nummatrix.h строка 285

```
285 { return{ n, m }; }
```

Граф вызова функции:



10.16.3.19 toIdentity()

template<typename T, size_t n, size_t m>
nummatrix<T, n, n>& nummatrix< T, n, m >::toIdentity ( ) [inline]

Установка матрицы в единичную

Template Parameters

T	тип данных
n	число строк и столбцов

Возвращает

ссылка на саму матрицу

См. определение в файле nummatrix.h строка 333

```
334    {
335        for (size_t i = 0; i < n; ++i)
336        for (size_t j = 0; j < m; ++j)
337            r[i*m + j] = (i == j) ? 1 : 0;
338        return *this;
339    }
```

10.16.3.20 toZero()

```
template<typename T, size_t n, size_t m>
nummatrix<T, n, m>& nummatrix< T, n, m >::toZero (
                T val = 0.0 )   [inline]
```

Установка всех компонент матрицы в константу (по умолчанию — нуль)

Template Parameters

T	тип данных
n	число строк
m	число столбцов

Аргументы

in	val	константа, значению которой приравниваются все компоненты матрицы (по умолчанию 0.0)

Возвращает

ссылка на саму матрицу

См. определение в файле nummatrix.h строка 320

```
321    {
322        for (size_t i = 0; i < n*m; ++i)
323            r[i] = val;
324        return *this;
325    }
```

10.16.3.21 transpose()

template<typename T, size_t n, size_t m>
nummatrix<T, m, n> nummatrix< T, n, m >::transpose ( ) [inline]

Транспонирование

См. определение в файле nummatrix.h строка 461

```
462    {
463        nummatrix<T, m, n> res;
464        for (size_t i = 0; i < n; ++i)
465        for (size_t j = 0; j < m; ++j)
466            res[j][i] = r[i*m + j];
467        return res;
468    }//Transpose(...)
```

10.16.4 Данные класса

10.16.4.1 r

template<typename T, size_t n, size_t m>
T nummatrix< T, n, m >::r[n *m]

Собственно, вектор с данными

См. определение в файле nummatrix.h строка 73

Объявления и описания членов класса находятся в файле:

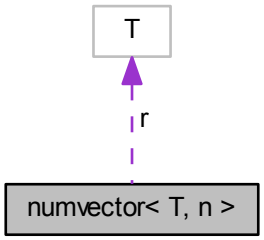
- numvector/nummatrix.h

10.17 Шаблон класса numvector< T, n >

Шаблонный класс, определяющий вектор фиксированной длины
Фактически представляет собой массив, для которого определено большое количество различных операций.
Для доступа к элементам массива используется оператор [].

#include <numvector.h>

Граф связей класса numvector< T, n >:



Открытые члены

- T & operator[] (size_t j)

  Оператор "[]" доступа к элементу
- const T & operator[] (size_t j) const

  Оператор "[]" доступа к элементу
- template<typename P >
  auto operator & (const numvector< P, n > &y) const -> typename std::remove_const< decltype(r[0] *y[0])>::type

  Оператор "&" скалярного умножения
- template<typename P >
  auto operator^ (const numvector< P, 3 > &y) const -> numvector< typename std::remove_const< decltype(r[1] *y[2])>::type, 3 >

  Оператор "^" векторного произведения
- template<typename P >
  auto operator^ (const numvector< P, 2 > &y) const -> typename std::remove_const< decltype(r[0] *y[1])>::type

  Оператор "^" вычисления третьей компоненты векторного произведения
- template<typename P >
  numvector< T, n > & operator*= (P c)

  Оператор "*=" домножения вектора на действительное число
- template<typename P >
  numvector< T, n > & operator/= (P c)

  Оператор "/=" деления вектора на действительное число
- template<typename P >
  numvector< T, n > & operator+= (const numvector< P, n > &y)

  Оператор "+=" прибавления другого вектора
- template<typename P >
  numvector< T, n > & operator-= (const numvector< P, n > &y)

  Оператор "-=" вычитания другого вектора
- template<typename P >
  auto operator+ (const numvector< P, n > &y) const -> numvector< typename std::remove_const< decltype(r[0]+y[0])>::type, n >

  Оператор "+" сложения двух векторов
- template<typename P >
  auto operator- (const numvector< P, n > &y) const -> numvector< typename std::remove_const< decltype(r[0] - y[0])>::type, n >

  Оператор "-" вычитания двух векторов
- template<typename P >
  auto operator* (const P c) const -> numvector< typename std::remove_const< decltype(r[0] *c)>::type, n >

  Оператор "*" умножения вектора на число (вектор слева, число справа)
- numvector< T, n > operator- () const

  Оператор "-" унарного минуса
- const numvector< T, n > & operator+ () const

  Оператор "+" унарного плюса
- template<typename P >
  bool operator== (const numvector< P, n > &y) const

  Оператор "==" логического равенства
- template<typename P >
  bool operator!= (const numvector< P, n > &y) const

  Перегрузка оператора "!=" логического неравенства
- size_t size () const

Вычисление размерности вектора (числа элементов в нем)

- auto norm1 () const -> typename std::remove_const< typename std::remove_reference< decltype(r[0])>::type >::type

  Вычисление 1-нормы вектора

- auto norminf () const -> typename std::remove_const< typename std::remove_reference< decltype(r[0])>::type >::type

  Вычисление inf-нормы вектора

- template<typename P = double>
  P length () const

  Вычисление 2-нормы (длины) вектора

- auto length2 () const -> typename std::remove_const< typename std::remove_reference< decltype(r[0])>::type >::type

  Вычисление квадрата нормы (длины) вектора

- template<typename P = double>
  auto unit (P newlen=1) const -> numvector< typename std::remove_const< decltype(r[0] *newlen)>::type, n >

  Вычисление орта вектора или вектора заданной длины, коллинеарного данному

- template<typename P = double>
  void normalize (P newlen=1.0)

  Нормирование вектора на заданную длину

- template<typename P >
  size_t member (const P &s) const

  Проверка вхождения элемента в вектор

- template<typename P >
  operator std::set< P > () const

  Приведение вектора к типу std::set.

- template<typename P >
  operator std::vector< P > () const

  Приведение вектора к типу std::vector.

- numvector< T, n > rotateLeft (size_t k) const

  "Вращение" вектора на несколько позиций влево

- numvector< T, 2 > kcross () const

  Геометрический поворот двумерного вектора на 90 градусов

- template<typename P = T>
  numvector< T, n > & toZero (P val=0)

  Установка всех компонент вектора в константу (по умолчанию — нуль)

- template<typename P >
  auto dist2To (const numvector< P, n > &y) -> typename std::remove_const< decltype(r[0] - y[0])>::type

  Вычисление квадрата расстояния до другой точки

- template<typename R = double, typename P >
  R distTo (const numvector< P, n > &y)

  Вычисление расстояния между двумя точками

- numvector ()

  Пустой конструктор

- template<typename P >
  numvector (const P c)

  Конструктор, инициализирующий весь вектор одной и той же константой

- numvector (const numvector< T, n > &vec)

  Нешаблонный конструктор копирования

- template<typename P >
  numvector (const numvector< P, n > &vec)

  Шаблонный конструктор копирования

- template<typename P >
  numvector (const std::vector< P > &vec)

  Конструктор инициализации с помощью std::vector.
- numvector (const std::initializer_list< T > &z)

  Нешаблонный конструктор инициализации списком
- template<typename P >
  numvector (const std::initializer_list< P > &z)

  Шаблонный конструктор инициализации списком
- template<typename P , size_t p>
  numvector (const numvector< P, p > &vec, T add=0)

  Явный конструктор инициализации вектором другой размерности
- numvector< T, n > operator= (double c)

  Оператор присваивания всем компонентам вектора одного и того же числа
- std::set< T > toSet () const

  Построение множества std::set на основе вектора
- std::vector< T > toVector () const

  Построение вектора std::vector на основе вектора

Защищенные данные

- T r [n]

  Собственно, вектор с данными

10.17.1 Подробное описание

template<typename T, size_t n>
class numvector< T, n >

Шаблонный класс, определяющий вектор фиксированной длины
Фактически представляет собой массив, для которого определено большое количество различных операций.
Для доступа к элементам массива используется оператор [].

Template Parameters

T	тип элементов вектора
n	длина вектора

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле numvector.h строка 90

10.17.2 Конструктор(ы)

10.17.2.1 numvector() [1/8]

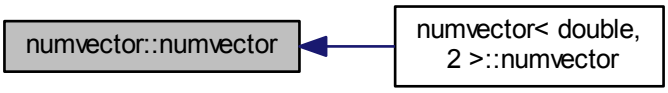
template<typename T, size_t n>
numvector< T, n >::numvector ( ) [inline]

Пустой конструктор

См. определение в файле numvector.h строка 554

554 { };

Граф вызова функции:



10.17.2.2 numvector() [2/8]

template<typename T, size_t n>
template<typename P >
numvector< T, n >::numvector (
const P c) [inline], [explicit]

Конструктор, инициализирующий весь вектор одной и той же константой

Template Parameters

P	тип данных присваиваемой константы

Аргументы

in	c	значение, которым инициализируются все компоненты вектора

См. определение в файле numvector.h строка 562

```
563    {
564        for (size_t i = 0; i < n; ++i)
565            r[i] = c;
566    }//numvector(...)
```

10.17.2.3 numvector() [3/8]

```
template<typename T, size_t n>
numvector< T, n >::numvector (
                const numvector< T, n > & vec )   [inline]
```

Нешаблонный конструктор копирования

Template Parameters

T	тип данных компонент вектора
n	длина вектора

Аргументы

in	vec	константная ссылка на копируемый вектор

См. определение в файле numvector.h строка 574

```
575    {
576        for (size_t i = 0; i < n; ++i)
577            r[i] = vec[i];
578    }//numvector(...)
```

10.17.2.4 numvector() [4/8]

```
template<typename T, size_t n>
template<typename P >
numvector< T, n >::numvector (
                const numvector< P, n > & vec )   [inline]
```

Шаблонный конструктор копирования

Template Parameters

P	тип данных копируемого вектора
n	длина вектора

Аргументы

in	vec	константная ссылка на копируемый вектор

См. определение в файле numvector.h строка 587

```
588     {
589         for (size_t i = 0; i < n; ++i)
590             r[i] = vec[i];
591     }//numvector(...)
```

10.17.2.5 numvector() [5/8]

```
template<typename T, size_t n>
template<typename P >
numvector< T, n >::numvector (
                const std::vector< P > & vec )   [inline]
```

Конструктор инициализации с помощью std::vector.

Template Parameters

P	тип данных инициализирующего std::vector
n	длина вектора

Аргументы

in	vec	константная ссылка на инициализирующий вектор

Предупреждения

при несовпадении длины кидает исключение

См. определение в файле numvector.h строка 602

```
603     {
604         if (vec.size() != n)
605             throw;
606         for (size_t i = 0; i < n; ++i)
607             r[i] = vec[i];
608     }//numvector(...)
```

10.17.2.6 numvector() [6/8]

```
template<typename T, size_t n>
numvector< T, n >::numvector (
            const std::initializer_list< T > & z )   [inline]
```

Нешаблонный конструктор инициализации списком

Template Parameters

T	тип данных

Аргументы

in	z	константная ссылка на список инициализации

Предупреждения

Длина списка инициализации проверяется, при несовпадении бросается исключение

См. определение в файле numvector.h строка 618

```
619     {
620         if (z.size() != n)
621             throw;
622         for (size_t i = 0; i < n; ++i)
623             r[i] = *(z.begin() + i);
624     }//numvector(...)
```

10.17.2.7 numvector() [7/8]

```
template<typename T, size_t n>
template<typename P >
numvector< T, n >::numvector (
            const std::initializer_list< P > & z )   [inline]
```

Шаблонный конструктор инициализации списком

Template Parameters

P	тип данных инициализирующего списка

Аргументы

in	z	константная ссылка на список инициализации

Предупреждения

Длина списка инициализации проверяется, при несовпадении бросается исключение

См. определение в файле numvector.h строка 634

```
{
    if (z.size() != n)
        throw;
    for (size_t i = 0; i < n; ++i)
        r[i] = *(z.begin() + i);
}//numvector(...)
```

10.17.2.8 numvector() [8/8]

```
template<typename T, size_t n>
template<typename P , size_t p>
numvector< T, n >::numvector (
                const numvector< P, p > & vec,
                T add = 0 )   [inline], [explicit]
```

Явный конструктор инициализации вектором другой размерности

- если новая размерность меньше старой — лишние элементы "отбрасываются"
- если новая размерность больше сиарой — новые элементы заполняются заданным элементом

Template Parameters

P	тип данных инициализирующего вектора
p	размерность инициализирующего вектора

Аргументы

in	vec	константная ссылка на инициализирующий вектор
in	add	элемент для заполнения при необходимости недостающих компонент (по умолчанию 0)

См. определение в файле numvector.h строка 654

```
{
    size_t minPN = (p<n) ? p : n;
    for (size_t i = 0; i < minPN; ++i)
        r[i] = vec[i];
    for (size_t i = minPN; i < n; ++i)
        r[i] = add;
}//numvector(...)
```

10.17.3 Методы

10.17.3.1 dist2To()

```
template<typename T, size_t n>
template<typename P >
auto numvector< T, n >::dist2To (
            const numvector< P, n > & y ) -> typename std::remove_const<decltype(r[0] - y[0])>::type    [inline]
```

Вычисление квадрата расстояния до другой точки

Template Parameters

P	тип данных второй точки

Аргументы

in	y	константная ссылка на радиус-вектор второй точки

Возвращает

квадрат расстояния между точками

См. определение в файле numvector.h строка 533

```
534    {
535        return (*this - y).length2();
536    }//dist2To(...)
```

10.17.3.2 distTo()

```
template<typename T, size_t n>
template<typename R = double, typename P >
R numvector< T, n >::distTo (
            const numvector< P, n > & y )    [inline]
```

Вычисление расстояния между двумя точками

Template Parameters

P	тип данных второй точки
R	тип данных результата (по умолчанию double)

Аргументы

in	y	константная ссылка на радиус-вектор второй точки

Возвращает

расстояние между точками

См. определение в файле numvector.h строка 546

```
547 {
548     R res = (*this-y) & (*this-y);
549     return sqrt(res);
550 }//distTo(...)
```

10.17.3.3 kcross()

template<typename T, size_t n>
numvector<T, 2> numvector< T, n >::kcross ( ) const [inline]

Геометрический поворот двумерного вектора на 90 градусов

Исходный вектор при этом не изменяется
Эквивалентно умножению слева на орт третьей оси, т.е. $\vec{k} \times \vec{r}$

Template Parameters

T	тип данных

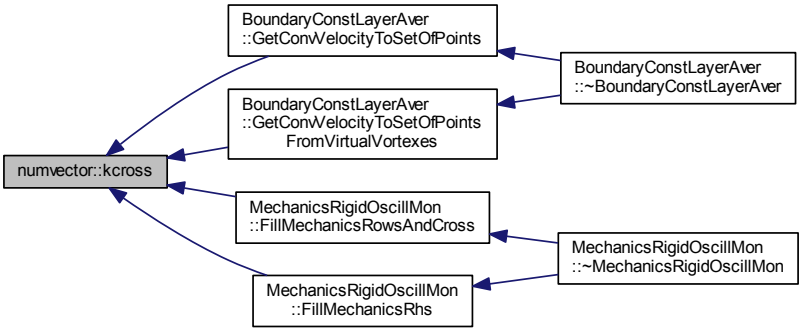
Возвращает

новый двумерный вектор, полученный поворотом исходного на 90 градусов

См. определение в файле numvector.h строка 502

```
503 {
504     numvector<T, 2> res;
505     res[0] = -r[1];
506     res[1] = r[0];
507     return res;
508 }//kcross()
```

Граф вызова функции:



10.17.3.4 length()

template<typename T, size_t n>
template<typename P = double>
P numvector< T, n >::length ( ) const [inline]

Вычисление 2-нормы (длины) вектора

Корень из скалярного квадрата вектора

Template Parameters

P	тип результата (по умолчанию double)

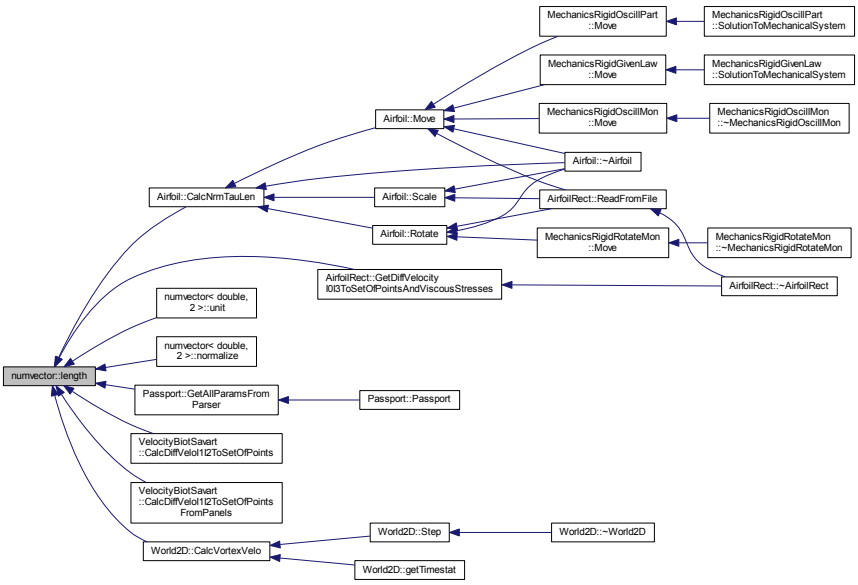
Возвращает

норма (длина) вектора

См. определение в файле numvector.h строка 382

```
383 {
384     P res = *this & *this;
385     return sqrt(res);
386 }//length()
```

Граф вызова функции:



10.17.3.5 length2()

template<typename T, size_t n>
auto numvector< T, n >::length2 ( ) const -> typename std::remove_const<typename std::remove_reference<decltype(r[0])>↩
::type>::type [inline]

Вычисление квадрата нормы (длины) вектора

Скалярный квадрат вектора

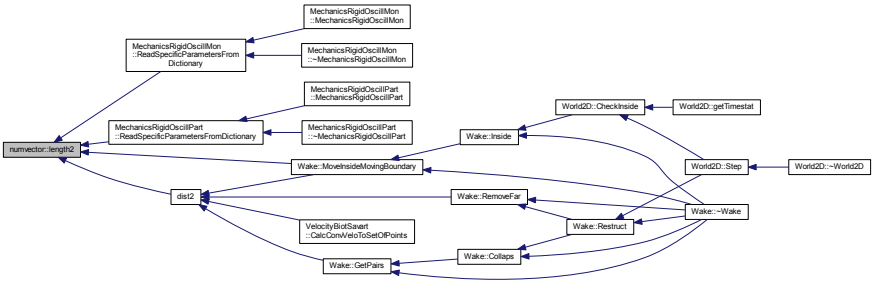
Возвращает

квадрат нормы (длины) вектора того же типа, что и компоненты вектора

См. определение в файле numvector.h строка 394

```
395 {
396     return (*this & *this);
397 }//length2()
```

Граф вызова функции:



10.17.3.6 member()

template<typename T, size_t n>
template<typename P >
size_t numvector< T, n >::member (
const P & s) const [inline]

Проверка вхождения элемента в вектор

Template Parameters

P	тип данных проверяемого элемента

Аргументы

in	s	проверяемый элемент

Возвращает

позиция первого вхождения элемента s; если не входит — возвращает (-1), приведенный к типу size_t

См. определение в файле numvector.h строка 439

```
440     {
441         for (size_t i = 0; i < n; ++i)
442         if (r[i] == s)
443             return i;
444
445         return static_cast<size_t>(-1);
446     }//member(...)
```

10.17.3.7 norm1()

```
template<typename T, size_t n>
auto numvector< T, n >::norm1 ( ) const -> typename std::remove_const<typename std::remove_reference<decltype(r[0])>↩
::type>::type    [inline]
```

Вычисление 1-нормы вектора

Сумма модулей компонент вектора

Возвращает

1-норма вектора

См. определение в файле numvector.h строка 348

```
349     {
350         typename std::remove_const<typename std::remove_reference<decltype(r[0])>::type>::type res = 0;
351         for (size_t i = 0; i < n; ++i)
352             res += abs(r[i]);
353         return res;
354     }//norm1()
```

10.17.3.8 normalize()

```
template<typename T, size_t n>
template<typename P = double>
void numvector< T, n >::normalize (
                P newlen = 1.0 )    [inline]
```

Нормирование вектора на заданную длину

Если в качестве новой длины указано отрицательное число — у вектора будет изменено направление

Template Parameters

P	тип числа, задающего длину вектора

Аргументы

in	newlen	новая длина вектора (по умолчанию 1.0)

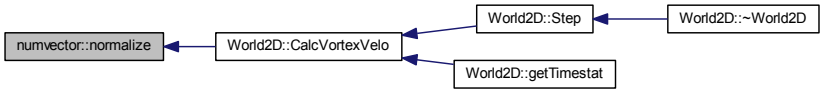
Предупреждения

Работает корректно только для векторов с компонентами типа float и double

См. определение в файле numvector.h строка 426

```
{
    auto ilen = static_cast<decltype(r[0] * newlen)>(newlen / this->length());
    *this *= ilen;
}//normalize(...)
```

Граф вызова функции:



10.17.3.9 norminf()

template<typename T, size_t n>
auto numvector< T, n >::norminf ( ) const -> typename std::remove_const<typename std::remove_reference<decltype(r[0])>↩
::type>::type [inline]

Вычисление inf-нормы вектора

Наибольшая по модулю компонента вектора

Возвращает

inf-норма вектора

См. определение в файле numvector.h строка 362

```
{
    typename std::remove_const<typename std::remove_reference<decltype(r[0])>::type>::type res = 0;
    for (size_t i = 0; i < n; ++i)
    {
        if (abs(r[i]) > res)
            res = abs(r[i]);
    }
    return res;
}//norminf()
```

10.17.3.10 operator &()

```
template<typename T, size_t n>
template<typename P >
auto numvector< T, n >::operator& (
            const numvector< P, n > & y ) const -> typename std::remove_const<decltype(r[0] * y[0])>::type    [inline]
```

Оператор "&" скалярного умножения

Template Parameters

P	тип данных компонент вектора - второго сомножителя
n	длина обоих векторов

Аргументы

in	y	константная ссылка на второй множитель

Возвращает

результат вычисления скалярного произведения, приведенный к нужному типу

См. определение в файле numvector.h строка 127

```
128    {
129        typename std::remove_const<decltype(r[0] * y[0])>::type res = 0;
130        for (size_t j = 0; j < n; ++j)
131            res += r[j] * y[j];
132        return res;
133    }//operator&(...)
```

10.17.3.11 operator std::set< P >()

```
template<typename T, size_t n>
template<typename P >
numvector< T, n >::operator std::set< P > ( ) const    [inline]
```

Приведение вектора к типу std::set.

Template Parameters

P	тип данных компонент множества

Возвращает

множество типа std::set, состоящее из тех же элементов, что исходный вектор

См. определение в файле numvector.h строка 454

```
455     {
456         std::set<P> newset;
457         for (size_t i = 0; i < n; ++i)
458             newset.insert(r[i]);
459         return newset;
460     }//toSet()
```

10.17.3.12 operator std::vector< P >()

template<typename T, size_t n>
template<typename P >
numvector< T, n >::operator std::vector< P > ( ) const [inline]

Приведение вектора к типу std::vector.

Template Parameters

P	тип данных компонент std::vector

Возвращает

вектор типа std::vector, состоящий из тех же элементов, что исходный вектор

См. определение в файле numvector.h строка 468

```
469     {
470         std::vector<P> vec;
471         vec.reserve(n);
472         for (size_t i = 0; i < n; ++i)
473             vec.push_back(r[i]);
474         return vec;
475     }
```

10.17.3.13 operator"!=()

template<typename T, size_t n>
template<typename P >
bool numvector< T, n >::operator!= (
const numvector< P, n > & y) const [inline]

Перегрузка оператора "!=" логического неравенства

Template Parameters

P	тип данных компонент вектора, с которым производится сравнение
n	длина вектора

Аргументы

in	y	константная ссылка на сравниваемый вектор

Возвращает

true, если векторы различаются, false в противном случае

См. определение в файле numvector.h строка 328

```
329     {
330         return !(*this == y);
331     }//operator!=(...)
```

10.17.3.14 operator∗()

```
template<typename T, size_t n>
template<typename P >
auto numvector< T, n >::operator∗ (
                    const P c ) const -> numvector<typename std::remove_const<decltype(r[0] ∗ c)>::type, n>    [inline]
```

Оператор "∗" умножения вектора на число (вектор слева, число справа)

Template Parameters

P	тип данных множителя
n	длина вектора

Аргументы

in	c	число-множитель

Возвращает

результат умножения вектора на число, приведенный к соответствующему типу

См. определение в файле numvector.h строка 271

```
272     {
273         numvector<typename std::remove_const<decltype(r[0] * c)>::type
    , n> res;
274         for (size_t i = 0; i < n; ++i)
275             res[i] = c * r[i];
276         return res;
277     }//operator*(...)
```

10.17.3.15 operator∗=()

```
template<typename T, size_t n>
template<typename P >
numvector<T, n>& numvector< T, n >::operator∗= (
                P c )  [inline]
```

Оператор "∗=" домножения вектора на действительное число

Template Parameters

T	тип данных компонент вектора
P	тип данных множителя
n	длина вектора

Аргументы

in	c	числовой множитель типа, приводимого к типу компонент вектора

Возвращает

ссылку на самого себя после домножения на число

См. определение в файле numvector.h строка 176

```
177     {
178         for (size_t i = 0; i < n; ++i)
179             r[i] *= c;
180         return *this;
181     }//operator*=(...)
```

10.17.3.16 operator+() [1/2]

```
template<typename T, size_t n>
template<typename P >
auto numvector< T, n >::operator+ (
                const numvector< P, n > & y ) const -> numvector<typename std::remove_const<decltype(r[0] + y[0])>↩
::type, n>  [inline]
```

Оператор "+" сложения двух векторов

Template Parameters

P	тип данных компонент вектора - второго слагаемого
n	длина вектора

Аргументы

in	y	константная ссылка на прибавляемый вектор

Возвращает

результат сложения двух векторов, приведенный к нужному типу

См. определение в файле numvector.h строка 239

```
240     {
241         numvector<typename std::remove_const<decltype(r[0] + y[0])>::type
    , n> res;
242         for (size_t i = 0; i < n; ++i)
243             res[i] = r[i] + y[i];
244         return res;
245     }//operator+(...)
```

10.17.3.17 operator+() [2/2]

```
template<typename T, size_t n>
const numvector<T, n>& numvector< T, n >::operator+ ( ) const    [inline]
```

Оператор "+" унарного плюса

Template Parameters

T	тип данных компонент вектора
n	длина вектора

Возвращает

константную ссылку на самого себя

См. определение в файле numvector.h строка 299

```
300     {
301         return *this;
302     }//operator+()
```

10.17.3.18 operator+=()

```
template<typename T, size_t n>
template<typename P >
numvector<T, n>& numvector< T, n >::operator+= (
                    const numvector< P, n > & y )    [inline]
```

Оператор "+=" прибавления другого вектора

Template Parameters

T	тип данных компонент вектора
P	тип данных компонент прибавляемого вектора
n	длина вектора

Аргументы

in	y	константная ссылка на прибавляемый вектор

Возвращает

ссылку на самого себя после сложения с другим вектором

См. определение в файле numvector.h строка 208

```
209     {
210         for (size_t i = 0; i < n; ++i)
211             r[i] += y[i];
212         return *this;
213     }//operator+=(...)
```

10.17.3.19 operator-() [1/2]

```
template<typename T, size_t n>
template<typename P >
auto numvector< T, n >::operator- (
                    const numvector< P, n > & y ) const -> numvector<typename std::remove_const<decltype(r[0] - y[0])>↩
::type, n>    [inline]
```

Оператор "-" вычитания двух векторов

Template Parameters

P	тип данных компонент вектора - вычитаемого
n	длина вектора

Аргументы

in	y	константная ссылка на вычитаемый вектор

Возвращает

результат вычитания двух векторов, приведенный к нужному типу

См. определение в файле numvector.h строка 255

```
256     {
257         numvector<typename std::remove_const<decltype(r[0] - y[0])>::type, n> res;
258         for (size_t i = 0; i < n; ++i)
259             res[i] = r[i] - y[i];
260         return res;
261     }//operator-(...)
```

10.17.3.20 operator-() [2/2]

template<typename T, size_t n>
numvector<T, n> numvector< T, n >::operator- ( ) const [inline]

Оператор "-" унарного минуса

Template Parameters

T	тип данных компонент вектора
n	длина вектора

Возвращает

противоположный вектор

См. определение в файле numvector.h строка 285

```
286     {
287         numvector<T, n> res;
288         for (size_t i = 0; i < n; ++i)
289             res[i] = -r[i];
290         return res;
291     }//operator-()
```

10.17.3.21 operator-=()

template<typename T, size_t n>
template<typename P >
numvector<T, n>& numvector< T, n >::operator-= (
const numvector< P, n > & y) [inline]

Оператор "-=" вычитания другого вектора

Template Parameters

T	тип данных компонент вектора
P	тип данных компонент вычитаемого вектора
n	длина вектора

Аргументы

in	y	константная ссылка на вычитаемый вектор

Возвращает

ссылка на самого себя после вычитания другого вектора

См. определение в файле numvector.h строка 224

```
225 {
226     for (size_t i = 0; i < n; ++i)
227         r[i] -= y[i];
228     return *this;
229 }//operator-=(...)
```

10.17.3.22 operator/=()

```
template<typename T, size_t n>
template<typename P >
numvector<T, n>& numvector< T, n >::operator/= (
                P c )  [inline]
```

Оператор "/=" деления вектора на действительное число

Template Parameters

T	тип данных компонент вектора
P	тип данных множителя
n	длина вектора

Аргументы

in	c	числовой делитель типа, приводимого к типу компонент вектора

Возвращает

ссылку на самого себя после деления на число

См. определение в файле numvector.h строка 192

```
193 {
194     for (size_t i = 0; i < n; ++i)
195         r[i] /= c;
196     return *this;
197 }//operator/=(...)
```

10.17.3.23 operator=()

```
template<typename T, size_t n>
numvector<T, n> numvector< T, n >::operator= (
                double c )   [inline]
```

Оператор присваивания всем компонентам вектора одного и того же числа

См. определение в файле numvector.h строка 670

```
671     {
672         return numvector<T, n>(c);
673     }//operator(...)
```

10.17.3.24 operator==()

```
template<typename T, size_t n>
template<typename P >
bool numvector< T, n >::operator== (
                const numvector< P, n > & y ) const   [inline]
```

Оператор "==" логического равенства

Template Parameters

P	тип данных компонент вектора, с которым производится сравнение
n	длина вектора

Аргументы

in	y	константная ссылка на сравниваемый вектор

Возвращает

true, если векторы одинаковые, false в противном случае

См. определение в файле numvector.h строка 312

```
313     {
314         for (size_t i = 0; i < n; ++i)
315         if (r[i] != y[i])
316             return false;
317         return true;
318     }//operator==(...)
```

10.17.3.25 operator[]() [1/2]

```
template<typename T, size_t n>
T& numvector< T, n >::operator[] (
                size_t j )   [inline]
```

Оператор "[]" доступа к элементу

Template Parameters

T	тип данных компонент вектора

Аргументы

in	j	номер элемента, к которому происходит обращение

Возвращает

Ссылка на элемент

См. определение в файле numvector.h строка 103

```
104     {
105         return r[j];
106     }//operator[](...)
```

10.17.3.26 operator[]() [2/2]

```
template<typename T, size_t n>
const T& numvector< T, n >::operator[] (
                    size_t j ) const    [inline]
```

Оператор "[]" доступа к элементу

Template Parameters

T	тип данных компонент вектора

Аргументы

in	j	номер элемента, к которому происходит обращение

Возвращает

Константная ссылка на элемент

См. определение в файле numvector.h строка 114

```
115     {
116         return r[j];
117     }//operator[](...)
```

10.17.3.27 operator^() [1/2]

```
template<typename T, size_t n>
template<typename P >
auto numvector< T, n >::operator^ (
            const numvector< P, 3 > & y ) const -> numvector<typename std::remove_const<decltype(r[1] * y[2])>↩
::type, 3>    [inline]
```

Оператор "^" векторного произведения

Определен только для трехмерных векторов

Template Parameters

P	тип данных компонент вектора - второго сомножителя

Аргументы

in	y	константная ссылка на второй множитель

Возвращает

результат вычисления векторного произведения, приведенный к нужному типу

См. определение в файле numvector.h строка 144

```
145     {
146         numvector<typename std::remove_const<decltype(r[1] * y[2])>::type
    , 3> vec;
147         vec[0] = r[1] * y[2] - r[2] * y[1];
148         vec[1] = r[2] * y[0] - r[0] * y[2];
149         vec[2] = r[0] * y[1] - r[1] * y[0];
150         return vec;
151     }//operator^(...)
```

10.17.3.28 operator^() [2/2]

```
template<typename T, size_t n>
template<typename P >
auto numvector< T, n >::operator^ (
            const numvector< P, 2 > & y ) const -> typename std::remove_const<decltype(r[0] * y[1])>::type    [inline]
```

Оператор "^" вычисления третьей компоненты векторного произведения

Определен только для двумерных векторов

Template Parameters

P	тип данных компонент вектора - второго множителя

Аргументы

in	y	константная ссылка на второй множитель

Возвращает

результат вычисления третьей компоненты векторного произведения двух двумерных векторов, приведенный к нужному типу

См. определение в файле numvector.h строка 162

```
163    {
164        return (r[0] * y[1] - r[1] * y[0]);
165    }//operator^(...)
```

10.17.3.29 rotateLeft()

```
template<typename T, size_t n>
numvector<T, n> numvector< T, n >::rotateLeft (
                size_t k ) const   [inline]
```

"Вращение" вектора на несколько позиций влево

Исходный вектор при этом не изменяется

Template Parameters

T	тип данных компонент вектора
n	длина вектора

Аргументы

in	k	количество позиций, на которые производится "вращение"

Возвращает

вектор, полученный "вращением" исходного на k позиций влево

См. определение в файле numvector.h строка 486

```
487    {
488        numvector<T, n> res;
489        for (size_t i = 0; i < n; ++i)
490            res[i] = r[(i + k) % n];
491        return res;
492    }//rotateLeft(...)
```

10.17.3.30 size()

template<typename T, size_t n>
size_t numvector< T, n >::size ( ) const [inline]

Вычисление размерности вектора (числа элементов в нем)

Возвращает

размерность вектора (число элементов в нем)

См. определение в файле numvector.h строка 337

```
338 {
339     return n;
340 }//size(...)
```

10.17.3.31 toSet()

template<typename T, size_t n>
std::set<T> numvector< T, n >::toSet ( ) const [inline]

Построение множества std::set на основе вектора

Возвращает

множество типа std::set, состоящее из тех же элементов, что исходный вектор

См. определение в файле numvector.h строка 679

```
680 {
681     //(deprecate: use implicit type conversion)
682
683     std::set<T> newset;
684     for (size_t i = 0; i < n; ++i)
685         newset.insert(r[i]);
686     return newset;
687 }//toSet()
```

10.17.3.32 toVector()

template<typename T, size_t n>
std::vector<T> numvector< T, n >::toVector ( ) const [inline]

Построение вектора std::vector на основе вектора

Возвращает

вектор типа std::vector, состоящий из тех же элементов, что исходный вектор

См. определение в файле numvector.h строка 693

```
694 {
695     //(deprecate: use implicit type conversion)
696
697     std::vector<T> vec;
698     vec.reserve(n);
699     for (size_t i = 0; i < n; ++i)
700         vec.push_back(r[i]);
701     return vec;
702 }
```

10.17.3.33 toZero()

```
template<typename T, size_t n>
template<typename P = T>
numvector<T, n>& numvector< T, n >::toZero (
                P val = 0 )  [inline]
```

Установка всех компонент вектора в константу (по умолчанию — нуль)

Template Parameters

T	тип данных компонент вектора
P	тип данных константы
n	длина вектора

Аргументы

in	val	константа, значению которой приравниваются все компоненты вектора (по умолчанию 0)

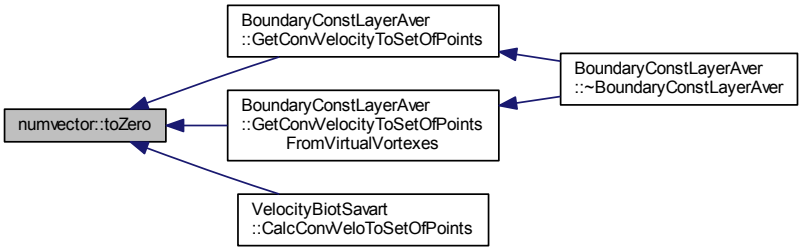
Возвращает

ссылка на сам вектор

См. определение в файле numvector.h строка 519

```
520    {
521        for (size_t i = 0; i < n; ++i)
522            r[i] = val;
523        return *this;
524    }
```

Граф вызова функции:



10.17.3.34 unit()

```
template<typename T, size_t n>
template<typename P = double>
auto numvector< T, n >::unit (
            P newlen = 1 ) const -> numvector<typename std::remove_const<decltype(r[0] * newlen)>::type, n>
[inline]
```

Вычисление орта вектора или вектора заданной длины, коллинеарного данному

Если в качестве новой длины указано отрицательное число — вектор будет противоположно направленным

Template Parameters

P	тип числа, задающего длину вектора

Аргументы

in	newlen	длина получаемого вектора (по умолчанию 1.0)

Возвращает

вектор, коллинеарный исходному, заданной длины (по умолчанию 1.0)

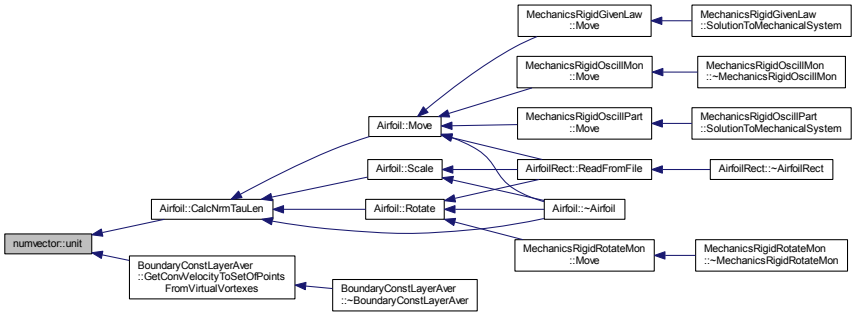
Предупреждения

для получения float-орта от вектора с компонентами типа float или целыми нужно явно указать параметр 1.0f

См. определение в файле numvector.h строка 410

```
411 {
412     auto ilen = static_cast<decltype(r[0] * newlen)>(newlen / this->length());
413     return (*this * ilen);
414 }//unit(...)
```

Граф вызова функции:



10.17.4 Данные класса

10.17.4.1 r

template<typename T, size_t n>
T numvector< T, n >::r[n] [protected]

Собственно, вектор с данными

См. определение в файле numvector.h строка 95

Объявления и описания членов класса находятся в файле:

- numvector/numvector.h

10.18 Класс Parallel

Класс, опеделяющий параметры исполнения задачи в параллельном MPI-режиме

#include <Parallel.h>

Открытые члены

- parProp SplitMPIone (size_t n, bool bcastAll=false) const

  Распределение задач по процессорам
- parProp SplitMPI (size_t n, bool bcastAll=false) const

  Распределение задач по процессорам

Открытые атрибуты

- MPI_Comm commWork

  Коммуникатор для решения конкретной задачи
- int myidWork

  Локальный номер процессора, решающего конкретную задачу
- int nProcWork

  Число процессоров, решающих конкретную задачу

10.18.1 Подробное описание

Класс, опеделяющий параметры исполнения задачи в параллельном MPI-режиме

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Parallel.h строка 82

10.18.2 Методы

10.18.2.1 SplitMPI()

```
parProp Parallel::SplitMPI (
            size_t n,
            bool bcastAll = false ) const
```

Распределение задач по процессорам

Аргументы

in	n	число распределяемых витков цикла
in	bcastAll	признак рассылки всей информации всем процессорам (по умолчанию false)

Возвращает

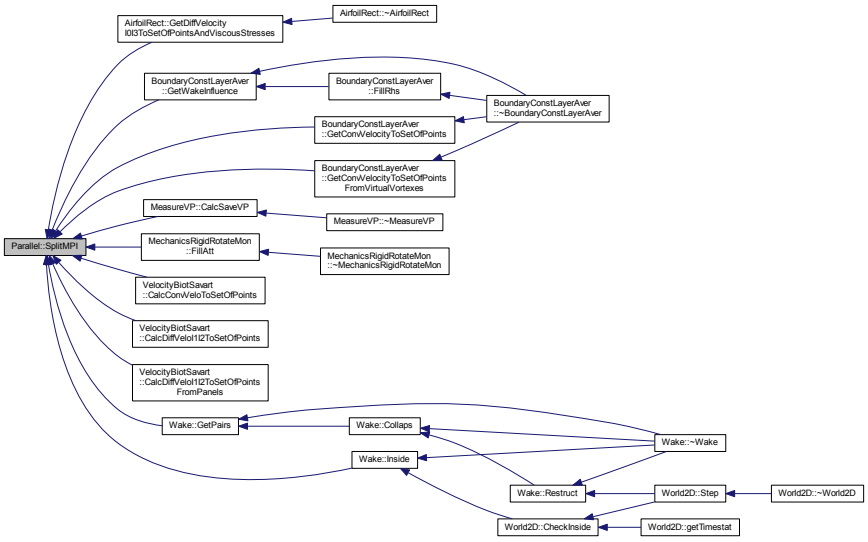
структуру типа parProp, заполненную для текущего процессора

См. определение в файле Parallel.cpp строка 85

```
86 {
87     parProp par;
88
89     par.totalLen = static_cast<int>(n);
90     MPI_Bcast(&par.totalLen, 1, MPI_INT, 0, commWork);
91
```

```
    if (myidWork == 0)
    {
        par.len.clear();
        par.disp.clear();

        int nPerP = static_cast<int>(n / nProcWork);

        for (int s = 0; s < nProcWork - 1; ++s)
         {
            par.len.push_back(nPerP);
            par.disp.push_back(s*nPerP);
         }

         par.len.push_back(static_cast<int>(n) - nPerP * (nProcWork - 1));
         par.disp.push_back(nPerP * (nProcWork - 1));
    }

    MPI_Scatter(par.len.data(), 1, MPI_INT, &par.myLen, 1, MPI_INT, 0, commWork);
    MPI_Scatter(par.disp.data(), 1, MPI_INT, &par.myDisp, 1, MPI_INT, 0, commWork);

    if (bcastAll)
    {
        if (myidWork != 0)
        {
        par.len.resize(nProcWork);
        par.disp.resize(nProcWork);
        }
        MPI_Bcast(par.len.data(),  nProcWork, MPI_INT, 0, commWork);
        MPI_Bcast(par.disp.data(), nProcWork, MPI_INT, 0, commWork);
    }

    return par;

}//SplitMPI(...)
```

Граф вызова функции:



10.18.2.2 SplitMPIone()

parProp Parallel::SplitMPIone (
size_t n,
bool bcastAll = false) const

Распределение задач по процессорам

Аргументы

in	n	число распределяемых витков цикла
in	bcastAll	признак рассылки всей информации всем процессорам (по умолчанию false)

Возвращает

структуру типа parProp, заполненную для текущего процессора

См. определение в файле Parallel.cpp строка 43

```
44 {
45     parProp par;
46
47     par.totalLen = static_cast<int>(n);
48     MPI_Bcast(&par.totalLen, 1, MPI_INT, 0, commWork);
49
50     if (myidWork == 0)
51     {
52         par.len.clear();
53         par.disp.clear();
54
55         par.len.push_back(static_cast<int>(n));
56         par.disp.push_back(0);
57
58         for (int s = 1; s < nProcWork; ++s)
59         {
60             par.len.push_back(0);
61             par.disp.push_back(static_cast<int>(n-1));
62         }
63     }
64
65     MPI_Scatter(par.len.data(), 1, MPI_INT, &par.myLen, 1, MPI_INT, 0,
   commWork);
66     MPI_Scatter(par.disp.data(), 1, MPI_INT, &par.myDisp, 1, MPI_INT, 0,
   commWork);
67
68     if (bcastAll)
69     {
70         if (myidWork != 0)
71         {
72             par.len.resize(nProcWork);
73             par.disp.resize(nProcWork);
74         }
75         MPI_Bcast(par.len.data(), nProcWork, MPI_INT, 0, commWork);
76         MPI_Bcast(par.disp.data(), nProcWork, MPI_INT, 0, commWork);
77     }
78
79     return par;
80
81 }//SplitMPIone(...)
```

10.18.3 Данные класса

10.18.3.1 commWork

MPI_Comm Parallel::commWork

Коммуникатор для решения конкретной задачи

См. определение в файле Parallel.h строка 86

10.18.3.2 myidWork

int Parallel::myidWork

Локальный номер процессора, решающего конкретную задачу

См. определение в файле Parallel.h строка 89

10.18.3.3 nProcWork

int Parallel::nProcWork

Число процессоров, решающих конкретную задачу

См. определение в файле Parallel.h строка 92

Объявления и описания членов классов находятся в файлах:

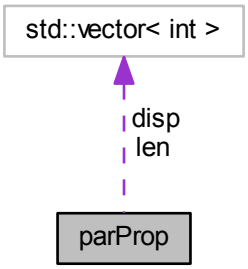
- Parallel/Parallel.h
- Parallel/Parallel.cpp

10.19 Структура parProp

Стрктура, содержащая параметры исполнения задачи в параллельном MPI-режиме

#include <Parallel.h>

Граф связей класса parProp:



Открытые атрибуты

- std::vector< int > len

  Список из чисел витков циклов, предназначенных для каждого процессора
- std::vector< int > disp

  Список, определяющий номер витка цикла, с которого должен начинать работу данный процессор
- int myLen

  Число витков, предназначенное текущему процессору
- int myDisp

  Индекс первого витка из числа витков, предназначенных текущему процессору
- int totalLen

  Общее число витков, разделенное между всеми процессорами

10.19.1 Подробное описание

Стрктура, содержащая параметры исполнения задачи в параллельном MPI-режиме

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Parallel.h строка 54

10.19.2 Данные класса

10.19.2.1 disp

std::vector<int> parProp::disp

Список, определяющий номер витка цикла, с которого должен начинать работу данный процессор

См. определение в файле Parallel.h строка 60

10.19.2.2 len

std::vector<int> parProp::len

Список из чисел витков циклов, предназначенных для каждого процессора

См. определение в файле Parallel.h строка 57

10.19.2.3 myDisp

int parProp::myDisp

Индекс первого витка из числа витков, предназначенных текущему процессору

См. определение в файле Parallel.h строка 66

10.19.2.4 myLen

int parProp::myLen

Число витков, предназначенное текущему процессору

См. определение в файле Parallel.h строка 63

10.19.2.5 totalLen

int parProp::totalLen

Общее число витков, разделенное между всеми процессорами

См. определение в файле Parallel.h строка 69

Объявления и описания членов структуры находятся в файле:

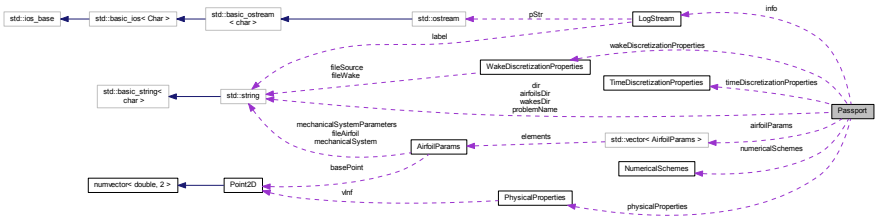
- Parallel/Parallel.h

10.20 Класс Passport

Класс, опеделяющий паспорт задачи

#include <Passport.h>

Граф связей класса Passport:



Открытые члены

- Passport (LogStream &infoStream, const std::string &_problemName, const size_t _problem↩ Number, const std::string &_filePassport, const std::string &_mechanics, const std::string &_↩ defaults, const std::string &_switchers, const std::vector< std::string > &vars)

  Конструктор
- virtual ~Passport ()

  Деструктор

Открытые атрибуты

- std::string dir

  Рабочий каталог задачи
- std::string problemName

  Название задачи
- size_t problemNumber

  Номер задачи
- std::string airfoilsDir

  Каталог с файлами профилей
- std::string wakesDir

  Каталог с файлами вихревых следов
- std::vector< AirfoilParams > airfoilParams

  Список структур с параметрами профилей
- PhysicalProperties physicalProperties

  Структура с физическими свойствами задачи
- TimeDiscretizationProperties timeDiscretizationProperties

  Структура с параметрами процесса интегрирования по времени
- WakeDiscretizationProperties wakeDiscretizationProperties

  Структура с параметрами дискретизации вихревого следа
- NumericalSchemes numericalSchemes

  Структура с используемыми численными схемами

Закрытые члены

- void GetAllParamsFromParser (std::istream &mainStream, std::istream &mechanicsStream, std↩ ::istream &defaultStream, std::istream &switcherStream, std::istream &varsStream)

  Считывание всех параметров расчета из соответствующих потоков
- void PrintAllParams ()

  Печать всех параметров расчета в поток логов

Закрытые данные

- LogStream info

  Поток для вывода логов и сообщений об ошибках

10.20.1 Подробное описание

Класс, опеделяющий паспорт задачи

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Passport.h строка 272

10.20.2 Конструктор(ы)

10.20.2.1 Passport()

```
Passport::Passport (
            LogStream & infoStream,
            const std::string & _problemName,
            const size_t _problemNumber,
            const std::string & _filePassport,
            const std::string & _mechanics,
            const std::string & _defaults,
            const std::string & _switchers,
            const std::vector< std::string > & vars )
```

Конструктор

Осуществляет чтение всех данных из соответствующих потоков, полностью инициализирует паспорт

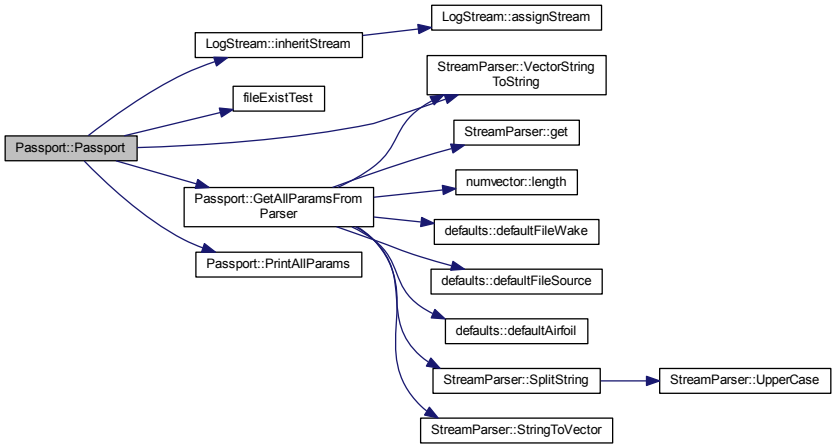
Аргументы

in,out	infoStream	базовый поток для вывода логов
in	_problemName	константная ссылка наназвание задачи
in	_problemNumber	номер (по счету) решаемой задачи
in	_filePassport	константная ссылка на файл (без пути) с паспортом задачи
in	_mechanics	константная ссылка на файл (с путем) со словарем механических систем
in	_defaults	константная ссылка на имя файла (с путем) с параметрами по умолчанию
in	_switchers	константная ссылка на имя файла (с путем) со значениями параметров-переключателей
in	vars	константная ссылка на список переменных, заданных в виде строк

См. определение в файле Passport.cpp строка 45

```
46 : problemName(_problemName), problemNumber(_problemNumber)
47 {
48     std::stringstream ss;
49     ss << "#" << problemNumber << " (" << problemName <<") passport";
50     info.inheritStream(infoStream, ss.str());
51
52     dir = "./" + problemName + "/";
53
54     std::string fileFullName = dir + _filePassport;
55     std::string mechanicsFileFullName = _mechanics;
56     std::string defaultsFileFullName = _defaults;
57     std::string switchersFileFullName = _switchers;
58
59     if (fileExistTest(fileFullName, info)
60         && fileExistTest(defaultsFileFullName, info)
61         && fileExistTest(switchersFileFullName, info)
62         && fileExistTest(mechanicsFileFullName, info))
63     {
64         std::string str = StreamParser::VectorStringToString(vars);
65         std::stringstream varsStream(str);
66
67         std::stringstream mainStream;
68         mainStream << Preprocessor(fileFullName).resultString;
69
70         std::stringstream mechanicsStream;
71         mechanicsStream << Preprocessor(mechanicsFileFullName).
    resultString;
72
73         std::stringstream defaultsStream;
74         defaultsStream << Preprocessor(defaultsFileFullName).
    resultString;
75
76         std::stringstream switchersStream;
77         switchersStream << Preprocessor(switchersFileFullName).
    resultString;
78
79         GetAllParamsFromParser(mainStream, mechanicsStream, defaultsStream,
    switchersStream, varsStream);
80
81         PrintAllParams();
82     }
83 }//Passport(...)
```

Граф вызовов:



10.20.2.2 ~Passport()

virtual Passport::~Passport ( ) [inline], [virtual]

Деструктор

См. определение в файле Passport.h строка 353

```
353 { };
```

10.20.3 Методы

10.20.3.1 GetAllParamsFromParser()

void Passport::GetAllParamsFromParser (
std::istream & mainStream,
std::istream & mechanicsStream,
std::istream & defaultStream,
std::istream & switcherStream,
std::istream & varsStream) [private]

Считывание всех параметров расчета из соответствующих потоков

Аргументы

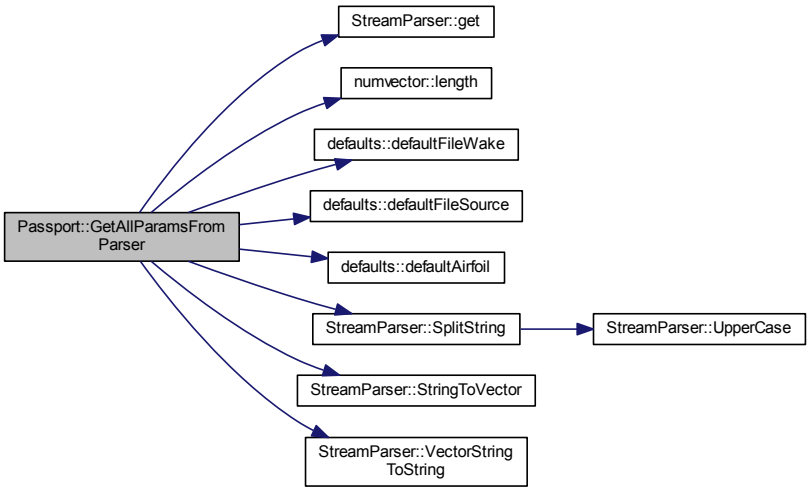
in	mainStream	ссылка на основной поток
in	mechanicsStream	ссылка на поток со словарем механических систем
in	defaultStream	ссылка на поток с параметрами по умолчанию
in	switcherStream	ссылка на поток со значениями параметров-переключателей
in	varsStream	ссылка на поток с параметрами конкретной задачи и переменными

См. определение в файле Passport.cpp строка 88

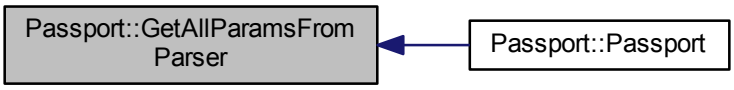
```
95 {
96
97 // 1. Разбор паспорта в целом
98
99     //создаем парсер и связываем его с нужным потоком
100    std::unique_ptr<StreamParser> parser;
101    parser.reset(new StreamParser(info, "parser", mainStream, defaultStream, switcherStream
   , varsStream));
102
103    //считываем общие парамеры
104    parser->get("rho", physicalProperties.rho);
105    parser->get("vInf", physicalProperties.vInf);
106    parser->get("vRef", physicalProperties.vRef, &
   defaults::defaultVRef);
107    if (physicalProperties.vRef == 0.0)
108    {
109        if (physicalProperties.vInf.length() == 0.0)
110        {
111            info('e') << "Reference velocity should be non-zero!" << std::endl;
112            exit(1);
113        }
114        physicalProperties.vRef = physicalProperties.
   vInf.length();
115    }
116
117    parser->get("timeAccel", physicalProperties.timeAccel, &
   defaults::defaultTimeAccel);
118    parser->get("nu", physicalProperties.nu);
119
120    parser->get("timeStart", timeDiscretizationProperties.
   timeStart, &defaults::defaultTimeStart);
121    parser->get("timeStop", timeDiscretizationProperties.
   timeStop);
122    parser->get("dt", timeDiscretizationProperties.
   dt);
123    parser->get("nameLength", timeDiscretizationProperties.
   nameLength, &defaults::defaultNameLength);
124    parser->get("saveTXT", timeDiscretizationProperties.
   saveTXT, &defaults::defaultSaveTXT);
125    parser->get("saveVTK", timeDiscretizationProperties.
   saveVTK, &defaults::defaultSaveVTK);
126    parser->get("saveVP",  timeDiscretizationProperties.
   saveVP,  &defaults::defaultSaveVP);
127
128
129    parser->get("eps", wakeDiscretizationProperties.
   eps);
130    wakeDiscretizationProperties.eps2 =
   wakeDiscretizationProperties.eps*
   wakeDiscretizationProperties.eps;
131    parser->get("epscol", wakeDiscretizationProperties.
   epscol);
132    parser->get("distFar", wakeDiscretizationProperties.
   distFar, &defaults::defaultDistFar);
133    parser->get("delta", wakeDiscretizationProperties.
   delta, &defaults::defaultDelta);
134    parser->get("vortexPerPanel", wakeDiscretizationProperties.
   vortexPerPanel, &defaults::defaultVortexPerPanel);
135
136    parser->get("linearSystemSolver", numericalSchemes.
   linearSystemSolver);
137    parser->get("velocityComputation", numericalSchemes.
   velocityComputation);
138    parser->get("wakeMotionIntegrator", numericalSchemes.
   wakeMotionIntegrator);
139
140    parser->get("airfoilsDir", airfoilsDir, &
   defaults::defaultAirfoilsDir);
141    parser->get("wakesDir",    wakesDir,    &
   defaults::defaultWakesDir);
142
143    parser->get("fileWake", wakeDiscretizationProperties.
   fileWake, &defaults::defaultFileWake);
144    parser->get("fileSource", wakeDiscretizationProperties.
   fileSource, &defaults::defaultFileSource);
145
146    std::vector<std::string> airfoil;
147    parser->get("airfoil", airfoil, &defaults::defaultAirfoil);
148
149    // 2. Разбор параметров профилей
150
151    //определяем число профилей и организуем цикл по ним
152    size_t nAirfoil = airfoil.size();
153    //*(defaults::defaultPinfo) << "Number of airfoils = " << nAirfoil << endl;
```

```
154     for (size_t i = 0; i < nAirfoil; ++i)
155     {
156         //делим имя файла + выражение в скобках на 2 подстроки
157         std::pair<std::string, std::string> airfoilLine =
    StreamParser::SplitString(info, airfoil[i], false);
158
159         AirfoilParams prm;
160         //первая подстрока - имя файла
161         prm.fileAirfoil = airfoilLine.first;
162
163         //вторую подстроку разделяем на вектор из строк по запятым, стоящим вне фигурных скобок
164         std::vector<std::string> vecAirfoilLineSecond =
    StreamParser::StringToVector(airfoilLine.second, '{', '}');
165
166         //создаем парсер и связываем его с параметрами профиля
167         std::stringstream aflStream(StreamParser::VectorStringToString(
    vecAirfoilLineSecond));
168         std::unique_ptr<StreamParser> parserAirfoil;
169
170         parserAirfoil.reset(new StreamParser(info, "airfoil parser", aflStream,
    defaultStream, switcherStream, varsStream));
171
172         //считываем нужные параметры с учетом default-значений
173         parserAirfoil->get("basePoint", prm.basePoint, &
    defaults::defaultBasePoint);
174         parserAirfoil->get("scale", prm.scale, &defaults::defaultScale);
175         parserAirfoil->get("angle", prm.angle, &defaults::defaultAngle);
176         prm.angle *= PI / 180.0;
177         parserAirfoil->get("panelsType", prm.panelsType, &
    defaults::defaultPanelsType);
178         parserAirfoil->get("inverse", prm.inverse, &
    defaults::defaultInverse);
179         parserAirfoil->get("boundaryConditionSatisfaction", prm.
    boundaryCondition, &defaults::defaultBoundaryCondition);
180         parserAirfoil->get("mechanicalSystem", prm.mechanicalSystem, &
    defaults::defaultMechanicalSystem);
181
182         if (prm.mechanicalSystem ==
    defaults::defaultMechanicalSystem)
183         {
184             prm.mechanicalSystemType = 0;
185             prm.mechanicalSystemParameters = "";
186         }
187         else
188         {
189             std::unique_ptr<StreamParser> parserMechanicsList;
190             std::unique_ptr<StreamParser> parserSwitchers;
191             parserMechanicsList.reset(new StreamParser(info, "mechanical parser",
    mechanicsStream, defaultStream, switcherStream, varsStream, { prm.mechanicalSystem }));
192             parserSwitchers.reset(new StreamParser(info, "switchers parser" ,switcherStream
    ));
193
194             std::string mechString;
195
196             parserMechanicsList->get(prm.mechanicalSystem, mechString);
197
198             //делим тип мех.системы + выражение в скобках (ее параметры) на 2 подстроки
199             std::pair<std::string, std::string> mechanicsLine =
    StreamParser::SplitString(info, mechString);
200
201             std::string mechTypeAlias = mechanicsLine.first;
202             parserSwitchers->get(mechTypeAlias, prm.mechanicalSystemType);
203
204             //вторую подстроку разделяем на вектор из строк по запятым, стоящим вне фигурных скобок
205             std::vector<std::string> vecMechLineSecond =
    StreamParser::StringToVector(mechanicsLine.second, '{', '}');
206             prm.mechanicalSystemParameters =
    StreamParser::VectorStringToString(vecMechLineSecond);
207         }
208
209         //отправляем считанные параметры профиля в структуру данных паспорта
210         airfoilParams.push_back(prm);
211
212     } //for i
213 }//GetAllParamsFromParser(...)
```

Граф вызовов:



Граф вызова функции:



10.20.3.2 PrintAllParams()

void Passport::PrintAllParams ( ) [private]

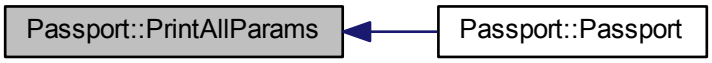
Печать всех параметров расчета в поток логов

См. определение в файле Passport.cpp строка 217

```
218 {
219     const std::string str = "passport info: ";
220
221     info('i') << "--- Passport info ---" << std::endl;
222     info('-') << "rho = " << physicalProperties.rho << std::endl;
223     info('-') << "vInf = " << physicalProperties.vInf << std::endl;
224     info('-') << "vRef = " << physicalProperties.vRef << std::endl;
225     info('-') << "nu = " << physicalProperties.nu << std::endl;
226     info('-') << "timeStart = " << timeDiscretizationProperties.
    timeStart << std::endl;
227     info('-') << "timeStop = " << timeDiscretizationProperties.
    timeStop << std::endl;
```

```
    info('-') << "dt = " << timeDiscretizationProperties.dt << std::endl;
    info('-') << "nameLength = " << timeDiscretizationProperties.nameLength << std::endl;
    info('-') << "saveTXT = " << timeDiscretizationProperties.saveTXT << std::endl;
    info('-') << "saveVTK = " << timeDiscretizationProperties.saveVTK << std::endl;
    info('-') << "saveVP = " << timeDiscretizationProperties.saveVP << std::endl;
    info('-') << "eps = " << wakeDiscretizationProperties.eps << std::endl;
    info('-') << "eps2 = " << wakeDiscretizationProperties.eps2 << std::endl;
    info('-') << "epscol = " << wakeDiscretizationProperties.epscol << std::endl;
    info('-') << "distFar = " << wakeDiscretizationProperties.distFar << std::endl;
    info('-') << "delta = " << wakeDiscretizationProperties.delta << std::endl;
    info('-') << "vortexPerPanel = " << wakeDiscretizationProperties.vortexPerPanel << std::endl;
    info('-') << "linearSystemSolver = " << numericalSchemes.linearSystemSolver << std::endl;
    info('-') << "velocityComputation = " << numericalSchemes.velocityComputation << std::endl;
    info('-') << "wakeMotionIntegrator = " << numericalSchemes.wakeMotionIntegrator << std::endl;

    info('-') << "airfoilsDir = " << airfoilsDir << std::endl;
    info('-') << "wakesDir = " << wakesDir << std::endl;

    info('-') << "number of airfoils = " << airfoilParams.size() << std::endl;
    for (size_t q = 0; q < airfoilParams.size(); ++q)
    {
        info('_') << "airfoil[" << q << "]_file = " << airfoilParams[q].fileAirfoil << std::endl;
        info('_') << "airfoil[" << q << "]_basePoint = " << airfoilParams[q].basePoint << std::endl;
        info('_') << "airfoil[" << q << "]_scale = " << airfoilParams[q].scale << std::endl;
        info('_') << "airfoil[" << q << "]_angle = " << airfoilParams[q].angle << std::endl;
        info('_') << "airfoil[" << q << "]_inverse = " << (airfoilParams[q].inverse ? "true": "false") << std::endl;
        info('_') << "airfoil[" << q << "]_panelType = " << airfoilParams[q].panelsType << std::endl;
        info('_') << "airfoil[" << q << "]_boundaryCondition = " << airfoilParams[q].boundaryCondition << std::endl;
        info('_') << "airfoil[" << q << "]_mechanicalSystem = " << airfoilParams[q].mechanicalSystem << std::endl;
    }

    info('-') << "fileWake = " << wakeDiscretizationProperties.fileWake << std::endl;
    info('-') << "fileSource = " << wakeDiscretizationProperties.fileSource << std::endl;
}
```

Граф вызова функции:



10.20.4 Данные класса

10.20.4.1 airfoilParams

std::vector<AirfoilParams> Passport::airfoilParams

Список структур с параметрами профилей

См. определение в файле Passport.h строка 314

10.20.4.2 airfoilsDir

std::string Passport::airfoilsDir

Каталог с файлами профилей

См. определение в файле Passport.h строка 308

10.20.4.3 dir

std::string Passport::dir

Рабочий каталог задачи

См. определение в файле Passport.h строка 299

10.20.4.4 info

LogStream Passport::info [mutable], [private]

Поток для вывода логов и сообщений об ошибках

См. определение в файле Passport.h строка 292

10.20.4.5 numericalSchemes

NumericalSchemes Passport::numericalSchemes

Структура с используемыми численными схемами

См. определение в файле Passport.h строка 326

10.20.4.6 physicalProperties

PhysicalProperties Passport::physicalProperties

Структура с физическими свойствами задачи

См. определение в файле Passport.h строка 317

10.20.4.7 problemName

std::string Passport::problemName

Название задачи

См. определение в файле Passport.h строка 302

10.20.4.8 problemNumber

size_t Passport::problemNumber

Номер задачи

См. определение в файле Passport.h строка 305

10.20.4.9 timeDiscretizationProperties

TimeDiscretizationProperties Passport::timeDiscretizationProperties

Структура с параметрами процесса интегрирования по времени

См. определение в файле Passport.h строка 320

10.20.4.10 wakeDiscretizationProperties

WakeDiscretizationProperties Passport::wakeDiscretizationProperties

Структура с параметрами дискретизации вихревого следа

См. определение в файле Passport.h строка 323

10.20.4.11 wakesDir

std::string Passport::wakesDir

Каталог с файлами вихревых следов

См. определение в файле Passport.h строка 311

Объявления и описания членов классов находятся в файлах:

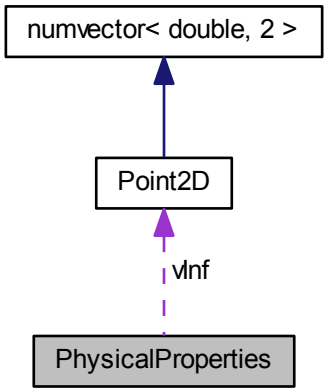
- Passport/Passport.h
- Passport/Passport.cpp

10.21 Структура PhysicalProperties

Структура, задающая физические свойства задачи

#include <Passport.h>

Граф связей класса PhysicalProperties:



Открытые члены

- double accelCft () const

  Функция-множитель, позволяющая моделировать разгон
- Point2D V0 () const

  Функция скорости набегающего потока с учетом разгона
- double getCurrTime () const

  Возвращает текуще время
- void setCurrTime (double t_) const

  Установка текущего времени
- void addCurrTime (double dt_) const

  Добавление шага к текущему времени

Открытые атрибуты

- double currTime

  Текущее время
- double rho

  Плотность потока
- Point2D vInf

  Скоростью набегающего потока
- double vRef

  Референсная скорость
- double timeAccel

  Время разгона
- double nu

  Коэффициент кинематической вязкости среды

10.21.1 Подробное описание

Структура, задающая физические свойства задачи

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Passport.h строка 69

10.21.2 Методы

10.21.2.1 accelCft()

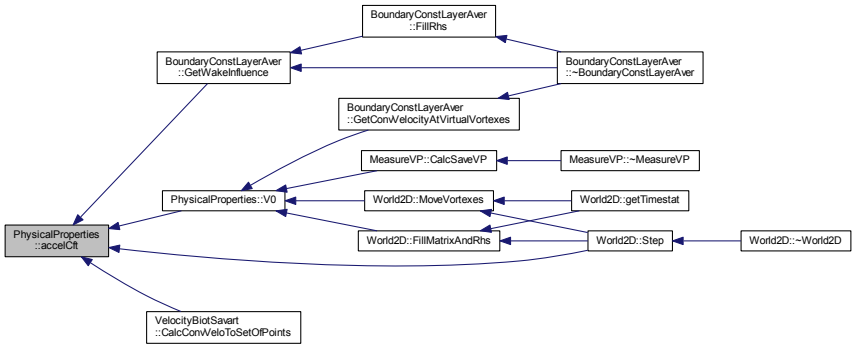
double PhysicalProperties::accelCft ( ) const [inline]

Функция-множитель, позволяющая моделировать разгон

См. определение в файле Passport.h строка 90

```
{
    return (currTime < timeAccel) ? (currTime / timeAccel) : 1.0;
}
```

Граф вызова функции:



10.21.2.2 addCurrTime()

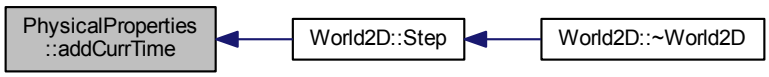
void PhysicalProperties::addCurrTime (
double dt_) const [inline]

Добавление шага к текущему времени

См. определение в файле Passport.h строка 118

```
{
    currTime += dt_;
}
```

Граф вызова функции:



10.21.2.3 getCurrTime()

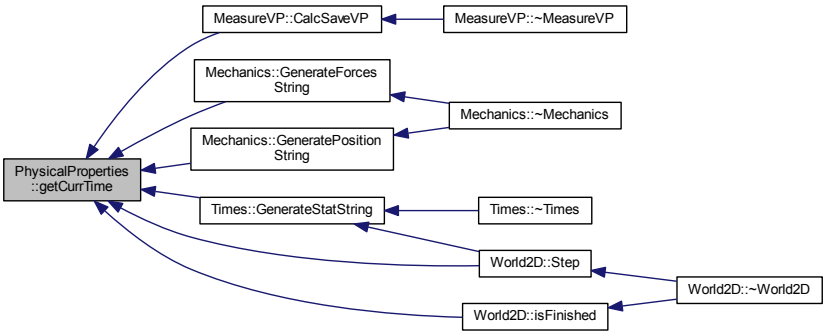
double PhysicalProperties::getCurrTime ( ) const [inline]

Возвращает текуще время

См. определение в файле Passport.h строка 106

```
107     {
108         return currTime;
109     }
```

Граф вызова функции:



10.21.2.4 setCurrTime()

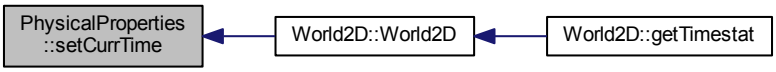
void PhysicalProperties::setCurrTime (
double t_) const [inline]

Установка текущего времени

См. определение в файле Passport.h строка 112

```
113     {
114         currTime = t_;
115     }
```

Граф вызова функции:



10.21.2.5 V0()

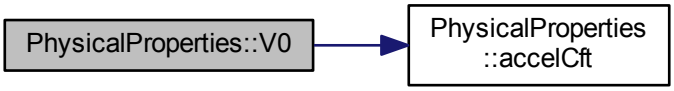
Point2D PhysicalProperties::V0 ( ) const [inline]

Функция скорости набегающего потока с учетом разгона

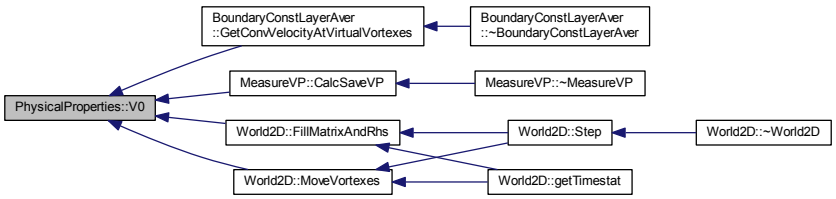
См. определение в файле Passport.h строка 96

```
97    {
98        return static_cast<Point2D>(accelCft()*vInf);
99        //return (currTime < timeAccel) ? static_cast<Point2D>(vInf * (currTime / timeAccel)) : vInf;
100    };
```

Граф вызовов:



Граф вызова функции:



10.21.3 Данные класса

10.21.3.1 currTime

double PhysicalProperties::currTime [mutable]

Текущее время

См. определение в файле Passport.h строка 74

10.21.3.2 nu

double PhysicalProperties::nu

Коэффициент кинематической вязкости среды

См. определение в файле Passport.h строка 100

10.21.3.3 rho

double PhysicalProperties::rho

Плотность потока

См. определение в файле Passport.h строка 78

10.21.3.4 timeAccel

double PhysicalProperties::timeAccel

Время разгона

См. определение в файле Passport.h строка 87

10.21.3.5 vInf

Point2D PhysicalProperties::vInf

Скоростью набегающего потока

См. определение в файле Passport.h строка 81

10.21.3.6 vRef

double PhysicalProperties::vRef

Референсная скорость

См. определение в файле Passport.h строка 84

Объявления и описания членов структуры находятся в файле:

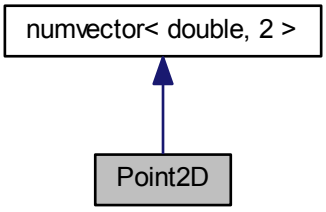
- Passport/Passport.h

10.22 Класс Point2D

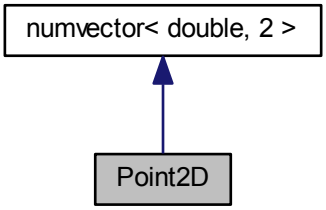
Класс, опеделяющий двумерный вектор

#include <Point2D.h>

Граф наследования:Point2D:



Граф связей класса Point2D:



Открытые члены

- Point2D ()

  Пустой конструктор
- Point2D (const numvector< double, 2 > &_r)

  Конструктор и приведение типа из numvector<double, 2>
- Point2D (const Point2D &_r)

  Конструктор копирования
- Point2D (const std::initializer_list< double > &z)

  Конструктор инициализации списком
- ~Point2D ()

  Деструктор
- Point2D rotated (const double angle=1.5707963267948966192313216916398) const

  Поворот вектора на произвольный угол против часовой стрелки (по умолчанию 90 градусов)

- double & operator[] (size_t j)

  Оператор "[]" доступа к элементу
- const double & operator[] (size_t j) const

  Оператор "[]" доступа к элементу
- auto operator& (const numvector< P, n > &y) const -> typename std::remove_const< decltype(r[0] *y[0])>::type

  Оператор "&" скалярного умножения
- auto operator^ (const numvector< P, 3 > &y) const -> numvector< typename std::remove_const< decltype(r[1] *y[2])>::type, 3 >

  Оператор "^" векторного произведения
- auto operator^ (const numvector< P, 2 > &y) const -> typename std::remove_const< decltype(r[0] *y[1])>::type

  Оператор "^" вычисления третьей компоненты векторного произведения
- numvector< double, n > & operator*= (P c)

  Оператор "*=" домножения вектора на действительное число
- numvector< double, n > & operator/= (P c)

  Оператор "/=" деления вектора на действительное число
- numvector< double, n > & operator+= (const numvector< P, n > &y)

  Оператор "+=" прибавления другого вектора
- numvector< double, n > & operator-= (const numvector< P, n > &y)

  Оператор "-=" вычитания другого вектора
- auto operator+ (const numvector< P, n > &y) const -> numvector< typename std::remove_const< decltype(r[0]+y[0])>::type, n >

  Оператор "+" сложения двух векторов
- const numvector< double, n > & operator+ () const

  Оператор "+" унарного плюса
- auto operator- (const numvector< P, n > &y) const -> numvector< typename std::remove_const< decltype(r[0] - y[0])>::type, n >

  Оператор "-" вычитания двух векторов
- numvector< double, n > operator- () const

  Оператор "-" унарного минуса
- auto operator* (const P c) const -> numvector< typename std::remove_const< decltype(r[0] *c)>::type, n >

  Оператор "*" умножения вектора на число (вектор слева, число справа)
- bool operator== (const numvector< P, n > &y) const

  Оператор "==" логического равенства
- bool operator!= (const numvector< P, n > &y) const

  Перегрузка оператора "!=" логического неравенства
- size_t size () const

  Вычисление размерности вектора (числа элементов в нем)
- auto norm1 () const -> typename std::remove_const< typename std::remove_reference< decltype(r[0])>::type >::type

  Вычисление 1-нормы вектора
- auto norminf () const -> typename std::remove_const< typename std::remove_reference< decltype(r[0])>::type >::type

  Вычисление inf-нормы вектора
- P length () const

  Вычисление 2-нормы (длины) вектора
- auto length2 () const -> typename std::remove_const< typename std::remove_reference< decltype(r[0])>::type >::type

  Вычисление квадрата нормы (длины) вектора

- auto unit (P newlen=1) const -> numvector< typename std::remove_const< decltype(r[0] *newlen)>::type, n >

  Вычисление орта вектора или вектора заданной длины, коллинеарного данному
- void normalize (P newlen=1.0)

  Нормирование вектора на заданную длину
- size_t member (const P &s) const

  Проверка вхождения элемента в вектор
- operator std::set< P > () const

  Приведение вектора к типу std::set.
- operator std::vector< P > () const

  Приведение вектора к типу std::vector.
- numvector< double, n > rotateLeft (size_t k) const

  "Вращение" вектора на несколько позиций влево
- numvector< double, 2 > kcross () const

  Геометрический поворот двумерного вектора на 90 градусов
- numvector< double, n > & toZero (P val=0)

  Установка всех компонент вектора в константу (по умолчанию — нуль)
- auto dist2To (const numvector< P, n > &y) -> typename std::remove_const< decltype(r[0] - y[0])>::type

  Вычисление квадрата расстояния до другой точки
- R distTo (const numvector< P, n > &y)

  Вычисление расстояния между двумя точками
- std::set< double > toSet () const

  Построение множества std::set на основе вектора
- std::vector< double > toVector () const

  Построение вектора std::vector на основе вектора

Открытые статические члены

- static void CreateMpiType ()

  Создание MPI-описателя типа

Статические открытые данные

- static MPI_Datatype mpiPoint2D

  MPI-описатель типа

Защищенные данные

- double r [n]

  Собственно, вектор с данными

10.22.1 Подробное описание

Класс, опеделяющий двумерный вектор

Наследуется от numvector<double, 2>, имеет дополнительные возможности:

- поворота на заданный угол против часовой стрелки;
- генерируется MPI-описатель для возможности его пересылки как единичного объекта.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Point2D.h строка 61

10.22.2 Конструктор(ы)

10.22.2.1 Point2D() [1/4]

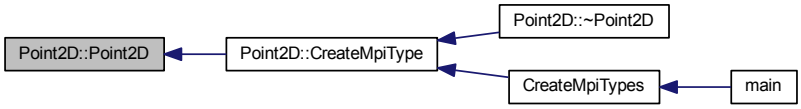
Point2D::Point2D ( ) [inline]

Пустой конструктор

См. определение в файле Point2D.h строка 69

```
69 { };
```

Граф вызова функции:



10.22.2.2 Point2D() [2/4]

Point2D::Point2D (
const numvector< double, 2 > & _r)

Конструктор и приведение типа из numvector<double, 2>

Аргументы

in	↩ _↩ r	константная ссылка на копируемый объект типа numvector<double, 2>

См. определение в файле Point2D.cpp строка 48

```
49 {
50     r[0] = _r[0];
51     r[1] = _r[1];
52 }//Point2D(...)
```

10.22.2.3 Point2D() [3/4]

```
Point2D::Point2D (
            const Point2D & _r )
```

Конструктор копирования

Аргументы

in	↩ _↩ r	константная ссылка на копируемый вектор

См. определение в файле Point2D.cpp строка 56

```
57 {
58     r[0] = _r[0];
59     r[1] = _r[1];
60 }//Point2D(...)
```

10.22.2.4 Point2D() [4/4]

```
Point2D::Point2D (
            const std::initializer_list< double > & z )
```

Конструктор инициализации списком

Аргументы

in	z	константная ссылка на список инициализации из чисел типа double

Предупреждения

Длина списка инициализации не проверяется, от него берутся только 2 первых элемента

См. определение в файле Point2D.cpp строка 64

```
65 {
66     for (size_t i = 0; i < 2; ++i)
67         r[i] = *(z.begin() + i);
68 }//Point2D(...)
```

10.22.2.5 ~Point2D()

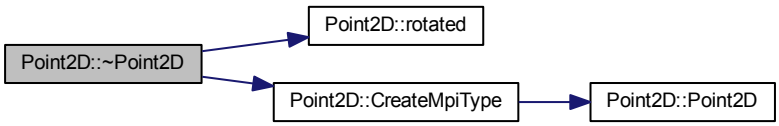
Point2D::~Point2D ( ) [inline]

Деструктор

См. определение в файле Point2D.h строка 90

```
90 { };
```

Граф вызовов:



10.22.3 Методы

10.22.3.1 CreateMpiType()

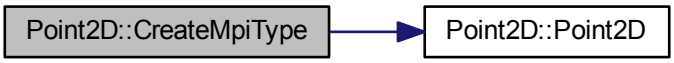
void Point2D::CreateMpiType ( ) [static]

Создание MPI-описателя типа

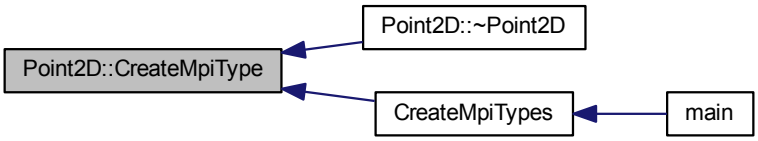
См. определение в файле Point2D.cpp строка 85

```
86 {
87     int          len[3] = { 1, 2, 1 };
88     MPI_Aint     pos[3] = { 0, offsetof(Point2D, r), sizeof(Point2D) };
89     MPI_Datatype typ[3] = { MPI_LB, MPI_DOUBLE, MPI_UB };
90
91     MPI_Type_create_struct(3, len, pos, typ, &mpiPoint2D);
92     MPI_Type_commit(&mpiPoint2D);
93 }//CreateMpiType()
```

Граф вызовов:



Граф вызова функции:



10.22.3.2 dist2To()

auto numvector< double , n >::dist2To (const numvector< P, n > & y)-> typename std::remove_const<decltype(r[0] - y[0])>::type [inline], [inherited]

Вычисление квадрата расстояния до другой точки

Template Parameters

P	тип данных второй точки

Аргументы

in	y	константная ссылка на радиус-вектор второй точки

Возвращает

квадрат расстояния между точками

См. определение в файле numvector.h строка 533

```
534    {
535        return (*this - y).length2();
536    }//dist2To(...)
```

10.22.3.3 distTo()

```
R numvector< double , n >::distTo (
                const numvector< P, n > & y )    [inline], [inherited]
```

Вычисление расстояния между двумя точками

Template Parameters

P	тип данных второй точки
R	тип данных результата (по умолчанию double)

Аргументы

in	y	константная ссылка на радиус-вектор второй точки

Возвращает

расстояние между точками

См. определение в файле numvector.h строка 546

```
547    {
548        R res = (*this-y) & (*this-y);
549        return sqrt(res);
550    }//distTo(...)
```

10.22.3.4 kcross()

numvector<double , 2> numvector< double , n >::kcross ( ) const [inline], [inherited]

Геометрический поворот двумерного вектора на 90 градусов

Исходный вектор при этом не изменяется
Эквивалентно умножению слева на орт третьей оси, т.е. $\vec{k} \times \vec{r}$

Template Parameters

T	тип данных

Возвращает

новый двумерный вектор, полученный поворотом исходного на 90 градусов

См. определение в файле numvector.h строка 502

```
503    {
504        numvector<T, 2> res;
505        res[0] = -r[1];
506        res[1] = r[0];
507        return res;
508    }//kcross()
```

10.22.3.5 length()

P numvector< double , n >::length ( ) const [inline], [inherited]

Вычисление 2-нормы (длины) вектора

Корень из скалярного квадрата вектора

Template Parameters

P	тип результата (по умолчанию double)

Возвращает

норма (длина) вектора

См. определение в файле numvector.h строка 382

```
383    {
384        P res = *this & *this;
385        return sqrt(res);
386    }//length()
```

10.22.3.6 length2()

auto numvector< double , n >::length2 ( ) const-> typename std::remove_const<typename std::remove_↩ reference<decltype(r[0])>::type>::type [inline], [inherited]

Вычисление квадрата нормы (длины) вектора

Скалярный квадрат вектора

Возвращает

квадрат нормы (длины) вектора того же типа, что и компоненты вектора

См. определение в файле numvector.h строка 394

```
395 {
396     return (*this & *this);
397 }//length2()
```

10.22.3.7 member()

size_t numvector< double , n >::member (
const P & s) const [inline], [inherited]

Проверка вхождения элемента в вектор

Template Parameters

P	тип данных проверяемого элемента

Аргументы

in	s	проверяемый элемент

Возвращает

позиция первого вхождения элемента s; если не входит — возвращает (-1), приведенный к типу size_t

См. определение в файле numvector.h строка 439

```
440 {
441     for (size_t i = 0; i < n; ++i)
442     if (r[i] == s)
443         return i;
444
445     return static_cast<size_t>(-1);
446 }//member(...)
```

10.22.3.8 norm1()

auto numvector< double , n >::norm1 ( ) const-> typename std::remove_const<typename std::remove_↩reference<decltype(r[0])>::type>::type [inline], [inherited]

Вычисление 1-нормы вектора

Сумма модулей компонент вектора

Возвращает

1-норма вектора

См. определение в файле numvector.h строка 348

```
349    {
350        typename std::remove_const<typename std::remove_reference<decltype(r[0])>::type>::type res = 0;
351        for (size_t i = 0; i < n; ++i)
352            res += abs(r[i]);
353        return res;
354    }//norm1()
```

10.22.3.9 normalize()

void numvector< double , n >::normalize (
P newlen = 1.0) [inline], [inherited]

Нормирование вектора на заданную длину

Если в качестве новой длины указано отрицательное число — у вектора будет изменено направление

Template Parameters

P	тип числа, задающего длину вектора

Аргументы

in	newlen	новая длина вектора (по умолчанию 1.0)

Предупреждения

Работает корректно только для векторов с компонентами типа float и double

См. определение в файле numvector.h строка 426

```
427    {
428        auto ilen = static_cast<decltype(r[0] * newlen)>(newlen / this->length());
429        *this *= ilen;
430    }//normalize(...)
```

10.22.3.10 norminf()

auto numvector< double , n >::norminf ( ) const-> typename std::remove_const<typename std::remove_↩ reference<decltype(r[0])>::type>::type [inline], [inherited]

Вычисление inf-нормы вектора

Наибольшая по модулю компонента вектора

Возвращает

inf-норма вектора

См. определение в файле numvector.h строка 362

```
{
    typename std::remove_const<typename std::remove_reference<decltype(r[0])>::type>::type res = 0;
    for (size_t i = 0; i < n; ++i)
    {
        if (abs(r[i]) > res)
            res = abs(r[i]);
    }
    return res;
}//norminf()
```

10.22.3.11 operator std::set< P >()

numvector< double , n >::operator std::set< P > ( ) const [inline], [inherited]

Приведение вектора к типу std::set.

Template Parameters

P	тип данных компонент множества

Возвращает

множество типа std::set, состоящее из тех же элементов, что исходный вектор

См. определение в файле numvector.h строка 454

```
{
    std::set<P> newset;
    for (size_t i = 0; i < n; ++i)
        newset.insert(r[i]);
    return newset;
}//toSet()
```

10.22.3.12 operator std::vector< P >()

numvector< double , n >::operator std::vector< P > ( ) const [inline], [inherited]

Приведение вектора к типу std::vector.

Template Parameters

P	тип данных компонент std::vector

Возвращает

вектор типа std::vector, состоящий из тех же элементов, что исходный вектор

См. определение в файле numvector.h строка 468

```
469 {
470     std::vector<P> vec;
471     vec.reserve(n);
472     for (size_t i = 0; i < n; ++i)
473         vec.push_back(r[i]);
474     return vec;
475 }
```

10.22.3.13 operator"!=()

bool numvector< double , n >::operator!= (
const numvector< P, n > & y) const [inline], [inherited]

Перегрузка оператора "!=" логического неравенства

Template Parameters

P	тип данных компонент вектора, с которым производится сравнение
n	длина вектора

Аргументы

in	y	константная ссылка на сравниваемый вектор

Возвращает

true, если векторы различаются, false в противном случае

См. определение в файле numvector.h строка 328

```
329 {
330     return !(*this == y);
331 }//operator!=(...)
```

10.22.3.14 operator&()

auto numvector< double , n >::operator& (
const numvector< P, n > & y) const-> typename std::remove_const<decltype(r[0] ∗ y[0])>::type [inline], [inherited]

Оператор "&" скалярного умножения

Template Parameters

P	тип данных компонент вектора - второго сомножителя
n	длина обоих векторов

Аргументы

in	y	константная ссылка на второй множитель

Возвращает

результат вычисления скалярного произведения, приведенный к нужному типу

См. определение в файле numvector.h строка 127

```
128     {
129         typename std::remove_const<decltype(r[0] * y[0])>::type res = 0;
130         for (size_t j = 0; j < n; ++j)
131             res += r[j] * y[j];
132         return res;
133     }//operator&(...)
```

10.22.3.15 operator∗()

auto numvector< double , n >::operator∗ (
const P c) const-> numvector<typename std::remove_const<decltype(r[0] ∗ c)>::type, n> [inline], [inherited]

Оператор "∗" умножения вектора на число (вектор слева, число справа)

Template Parameters

P	тип данных множителя
n	длина вектора

Аргументы

in	c	число-множитель

Возвращает

результат умножения вектора на число, приведенный к соответствующему типу

См. определение в файле numvector.h строка 271

```
272     {
273         numvector<typename std::remove_const<decltype(r[0] * c)>::type
    , n> res;
274         for (size_t i = 0; i < n; ++i)
275             res[i] = c * r[i];
276         return res;
277     }//operator*(...)
```

10.22.3.16 operator∗=()

numvector<double , n>& numvector< double , n >::operator∗= (
P c) [inline], [inherited]

Оператор "∗=" домножения вектора на действительное число

Template Parameters

T	тип данных компонент вектора
P	тип данных множителя
n	длина вектора

Аргументы

in	c	числовой множитель типа, приводимого к типу компонент вектора

Возвращает

ссылку на самого себя после домножения на число

См. определение в файле numvector.h строка 176

```
177     {
178         for (size_t i = 0; i < n; ++i)
179             r[i] *= c;
180         return *this;
181     }//operator*=(...)
```

10.22.3.17 operator+() [1/2]

auto numvector< double , n >::operator+ (
const numvector< P, n > & y) const-> numvector<typename std::remove_const<decltype(r[0] + y[0])>↩
::type, n> [inline], [inherited]

Оператор "+" сложения двух векторов

Template Parameters

P	тип данных компонент вектора - второго слагаемого
n	длина вектора

Аргументы

in	y	константная ссылка на прибавляемый вектор

Возвращает

результат сложения двух векторов, приведенный к нужному типу

См. определение в файле numvector.h строка 239

```
240     {
241         numvector<typename std::remove_const<decltype(r[0] + y[0])>::type
    , n> res;
242         for (size_t i = 0; i < n; ++i)
243             res[i] = r[i] + y[i];
244         return res;
245     }//operator+(...)
```

10.22.3.18 operator+() [2/2]

const numvector<double , n>& numvector< double , n >::operator+ ( ) const [inline], [inherited]

Оператор "+" унарного плюса

Template Parameters

T	тип данных компонент вектора
n	длина вектора

Возвращает

константную ссылку на самого себя

См. определение в файле numvector.h строка 299

```
300     {
301         return *this;
302     }//operator+()
```

10.22.3.19 operator+=()

numvector<double , n>& numvector< double , n >::operator+= (
const numvector< P, n > & y) [inline], [inherited]

Оператор "+=" прибавления другого вектора

Template Parameters

T	тип данных компонент вектора
P	тип данных компонент прибавляемого вектора
n	длина вектора

Аргументы

in	y	константная ссылка на прибавляемый вектор

Возвращает

ссылку на самого себя после сложения с другим вектором

См. определение в файле numvector.h строка 208

```
{
    for (size_t i = 0; i < n; ++i)
        r[i] += y[i];
    return *this;
}//operator+=(...)
```

10.22.3.20 operator-() [1/2]

auto numvector< double , n >::operator- (
const numvector< P, n > & y) const-> numvector<typename std::remove_const<decltype(r[0] - y[0])>↩
::type, n> [inline], [inherited]

Оператор "-" вычитания двух векторов

Template Parameters

P	тип данных компонент вектора - вычитаемого
n	длина вектора

Аргументы

in	y	константная ссылка на вычитаемый вектор

Возвращает

результат вычитания двух векторов, приведенный к нужному типу

См. определение в файле numvector.h строка 255

```
{
```

```
257         numvector<typename std::remove_const<decltype(r[0] - y[0])>::type, n> res;
258         for (size_t i = 0; i < n; ++i)
259             res[i] = r[i] - y[i];
260         return res;
261     }//operator-(...)
```

10.22.3.21 operator-() [2/2]

numvector<double , n> numvector< double , n >::operator- ( ) const [inline], [inherited]

Оператор "-" унарного минуса

Template Parameters

T	тип данных компонент вектора
n	длина вектора

Возвращает

противоположный вектор

См. определение в файле numvector.h строка 285

```
286     {
287         numvector<T, n> res;
288         for (size_t i = 0; i < n; ++i)
289             res[i] = -r[i];
290         return res;
291     }//operator-()
```

10.22.3.22 operator-=()

numvector<double , n>& numvector< double , n >::operator-= (
const numvector< P, n > & y) [inline], [inherited]

Оператор "-=" вычитания другого вектора

Template Parameters

T	тип данных компонент вектора
P	тип данных компонент вычитаемого вектора
n	длина вектора

Аргументы

in	y	константная ссылка на вычитаемый вектор

Возвращает

ссылка на самого себя после вычитания другого вектора

См. определение в файле numvector.h строка 224

```
225    {
226        for (size_t i = 0; i < n; ++i)
227            r[i] -= y[i];
228        return *this;
229    }//operator-=(...)
```

10.22.3.23 operator/=()

numvector<double , n>& numvector< double , n >::operator/= (
P c) [inline], [inherited]

Оператор "/=" деления вектора на действительное число

Template Parameters

T	тип данных компонент вектора
P	тип данных множителя
n	длина вектора

Аргументы

in	c	числовой делитель типа, приводимого к типу компонент вектора

Возвращает

ссылку на самого себя после деления на число

См. определение в файле numvector.h строка 192

```
193    {
194        for (size_t i = 0; i < n; ++i)
195            r[i] /= c;
196        return *this;
197    }//operator/=(...)
```

10.22.3.24 operator==()

bool numvector< double , n >::operator== (
const numvector< P, n > & y) const [inline], [inherited]

Оператор "==" логического равенства

Template Parameters

P	тип данных компонент вектора, с которым производится сравнение
n	длина вектора

Аргументы

in	y	константная ссылка на сравниваемый вектор

Возвращает

true, если векторы одинаковые, false в противном случае

См. определение в файле numvector.h строка 312

```
313 {
314     for (size_t i = 0; i < n; ++i)
315     if (r[i] != y[i])
316         return false;
317     return true;
318 }//operator==(...)
```

10.22.3.25 operator[]() [1/2]

```
double & numvector< double , n >::operator[] (
                size_t j )   [inline], [inherited]
```

Оператор "[]" доступа к элементу

Template Parameters

T	тип данных компонент вектора

Аргументы

in	j	номер элемента, к которому происходит обращение

Возвращает

Ссылка на элемент

См. определение в файле numvector.h строка 103

```
104 {
105     return r[j];
106 }//operator[](...)
```

10.22.3.26 operator[]() [2/2]

```
const double & numvector< double , n >::operator[] (
            size_t j ) const   [inline], [inherited]
```

Оператор "[]" доступа к элементу

Template Parameters

T	тип данных компонент вектора

Аргументы

in	j	номер элемента, к которому происходит обращение

Возвращает

Константная ссылка на элемент

См. определение в файле numvector.h строка 114

```
{
    return r[j];
}//operator[](...)
```

10.22.3.27 operator^() [1/2]

```
auto numvector< double , n >::operator^ (
            const numvector< P, 3 > & y ) const-> numvector<typename std::remove_const<decltype(r[1] * y[2])>↩
::type, 3>   [inline], [inherited]
```

Оператор "^" векторного произведения

Определен только для трехмерных векторов

Template Parameters

P	тип данных компонент вектора - второго сомножителя

Аргументы

in	y	константная ссылка на второй множитель

Возвращает

результат вычисления векторного произведения, приведенный к нужному типу

См. определение в файле numvector.h строка 144

```
145     {
146         numvector<typename std::remove_const<decltype(r[1] * y[2])>::type
    , 3> vec;
147         vec[0] = r[1] * y[2] - r[2] * y[1];
148         vec[1] = r[2] * y[0] - r[0] * y[2];
149         vec[2] = r[0] * y[1] - r[1] * y[0];
150         return vec;
151     }//operator^(...)
```

10.22.3.28 operator^() [2/2]

auto numvector< double , n >::operator^ (
const numvector< P, 2 > & y) const-> typename std::remove_const<decltype(r[0] ∗ y[1])>::type [inline], [inherited]

Оператор "^" вычисления третьей компоненты векторного произведения

Определен только для двумерных векторов

Template Parameters

P	тип данных компонент вектора - второго множителя

Аргументы

in	y	константная ссылка на второй множитель

Возвращает

результат вычисления третьей компоненты векторного произведения двух двумерных векторов, приведенный к нужному типу

См. определение в файле numvector.h строка 162

```
163     {
164         return (r[0] * y[1] - r[1] * y[0]);
165     }//operator^(...)
```

10.22.3.29 rotated()

Point2D Point2D::rotated (
const double angle = 1.5707963267948966192313216916398) const

Поворот вектора на произвольный угол против часовой стрелки (по умолчанию 90 градусов)

Аргументы

in	angle	угол поворота в радианах (по умолчанию $\frac{\pi}{2}$)

Возвращает

новый вектор, полученный поворотом старого

См. определение в файле Point2D.cpp строка 72

```
73 {
74     Point2D res;
75     double cosa = cos(angle);
76     double sina = sin(angle);
77
78     res[0] = r[0] * cosa - r[1] * sina;
79     res[1] = r[0] * sina + r[1] * cosa;
80     return res;
81 }//rotated(...)
```

Граф вызова функции:

10.22.3.30 rotateLeft()

numvector<double , n> numvector< double , n >::rotateLeft (
size_t k) const [inline], [inherited]

"Вращение" вектора на несколько позиций влево

Исходный вектор при этом не изменяется

Template Parameters

T	тип данных компонент вектора
n	длина вектора

Аргументы

in	k	количество позиций, на которые производится "вращение"

Возвращает

вектор, полученный "вращением" исходного на k позиций влево

См. определение в файле numvector.h строка 486

```
487    {
488        numvector<T, n> res;
489        for (size_t i = 0; i < n; ++i)
490            res[i] = r[(i + k) % n];
491        return res;
492    }//rotateLeft(...)
```

10.22.3.31 size()

size_t numvector< double , n >::size ( ) const [inline], [inherited]

Вычисление размерности вектора (числа элементов в нем)

Возвращает

размерность вектора (число элементов в нем)

См. определение в файле numvector.h строка 337

```
338    {
339        return n;
340    }//size(...)
```

10.22.3.32 toSet()

std::set<double > numvector< double , n >::toSet ( ) const [inline], [inherited]

Построение множества std::set на основе вектора

Возвращает

множество типа std::set, состоящее из тех же элементов, что исходный вектор

См. определение в файле numvector.h строка 679

```
680    {
681        //(deprecate: use implicit type conversion)
682
683        std::set<T> newset;
684        for (size_t i = 0; i < n; ++i)
685            newset.insert(r[i]);
686        return newset;
687    }//toSet()
```

10.22.3.33 toVector()

std::vector<double > numvector< double , n >::toVector ( ) const [inline], [inherited]

Построение вектора std::vector на основе вектора

Возвращает

вектор типа std::vector, состоящий из тех же элементов, что исходный вектор

См. определение в файле numvector.h строка 693

```
694 {
695     //(deprecate: use implicit type conversion)
696
697     std::vector<T> vec;
698     vec.reserve(n);
699     for (size_t i = 0; i < n; ++i)
700         vec.push_back(r[i]);
701     return vec;
702 }
```

10.22.3.34 toZero()

numvector<double , n>& numvector< double , n >::toZero (
P val = 0) [inline], [inherited]

Установка всех компонент вектора в константу (по умолчанию — нуль)

Template Parameters

T	тип данных компонент вектора
P	тип данных константы
n	длина вектора

Аргументы

in	val	константа, значению которой приравниваются все компоненты вектора (по умолчанию 0)

Возвращает

ссылка на сам вектор

См. определение в файле numvector.h строка 519

```
520 {
521     for (size_t i = 0; i < n; ++i)
522         r[i] = val;
523     return *this;
524 }
```

10.22.3.35 unit()

auto numvector< double , n >::unit (
P newlen = 1) const-> numvector<typename std::remove_const<decltype(r[0] * newlen)>::type, n>
[inline], [inherited]

Вычисление орта вектора или вектора заданной длины, коллинеарного данному

Если в качестве новой длины указано отрицательное число — вектор будет противоположно направленным

Template Parameters

P	тип числа, задающего длину вектора

Аргументы

in	newlen	длина получаемого вектора (по умолчанию 1.0)

Возвращает

вектор, коллинеарный исходному, заданной длины (по умолчанию 1.0)

Предупреждения

для получения float-орта от вектора с компонентами типа float или целыми нужно явно указать параметр 1.0f

См. определение в файле numvector.h строка 410

```
411 {
412     auto ilen = static_cast<decltype(r[0] * newlen)>(newlen / this->length());
413     return (*this * ilen);
414 }//unit(...)
```

10.22.4 Данные класса

10.22.4.1 mpiPoint2D

MPI_Datatype Point2D::mpiPoint2D [static]

MPI-описатель типа

См. определение в файле Point2D.h строка 66

10.22.4.2 r

double numvector< double , n >::r[n] [protected], [inherited]

Собственно, вектор с данными

См. определение в файле numvector.h строка 95

Объявления и описания членов классов находятся в файлах:

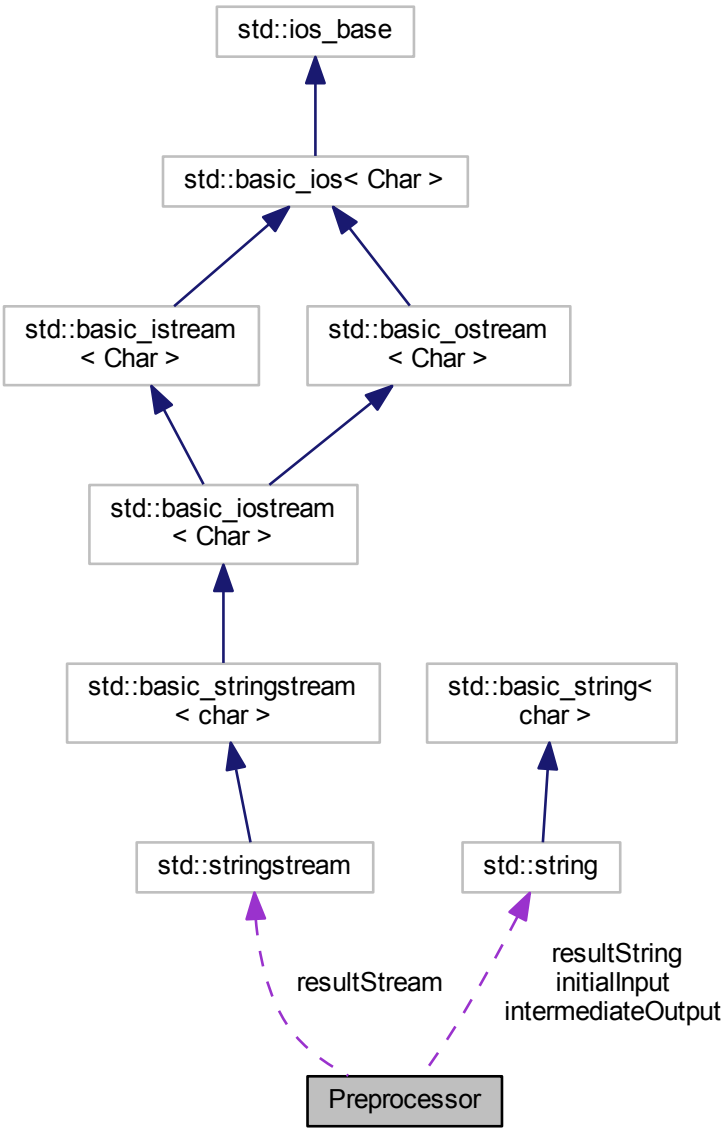
- Point2D/Point2D.h
- Point2D/Point2D.cpp

10.23 Класс Preprocessor

Класс, позволяющий выполнять предварительную обработку файлов

#include <Preprocessor.h>

Граф связей класса Preprocessor:



Открытые члены

- Preprocessor (const std::string &fileName)

  Конструктор, принимающий на вход имя обрабатываемого файла
- ~Preprocessor ()

  Деструктор

Открытые атрибуты

- std::string initialInput

Строка, содержащая исходный файл в первоначальном виде

- std::string intermediateOutput

  Строка, содержащая результат промежуточной обработки файла

- std::string resultString

  Строка, содержащая окончательный результат обработки файла

- std::stringstream resultStream

Закрытые члены

- void normalParser (char ch, std::string &str)

  Обработчик символа в режиме обычного состояния парсера

- void inStringParser (char ch, std::string &str)

  Обработчик символа в режиме парсера строки (внутри двойных кавычек)

- void inCharParser (char ch, std::string &str)

  Обработчик символа в режиме парсера символа (внутри одинарных кавычек)

- void afterSlashParser (char ch, std::string &str)

  Обработчик символа в режиме парсера выражения после слэша

- void inInlineCommentParser (char ch, std::string &str)

  Обработчик символа в режиме парсера однострочного комментария (после //)

- void inMultilineCommentParser (char ch, std::string &str)

  Обработчик символа в режиме парсера многострочного комментария

- char processSymbol (char ch)

  Базовая функция обработки символа В зависимости от входого символа возвращает на выход:

Закрытые данные

- bool inStringLastEscape

  Признак встречи слэша внутри строки (внутри двойных кавычек)

- bool inCharLastEscape

  Признак встречи слэша внутри символа (внутри одинарных кавычек)

- bool inInlineLastEscape

  Признак встречи слэша внутри однострочного комментария (после //)

- bool inMultilineLastStar

  Признак встречи звездочки внутри многострочного комментария (после '/*')

- void(Preprocessor::* currentParser)(char ch, std::string &str)

  Опрededение currentParser как указателя на функцию-члена класса

10.23.1 Подробное описание

Класс, позволяющий выполнять предварительную обработку файлов

Выполняет следующие действия:

- Исключает из файла однострочные и многостройчные комментарии в стиле C/C++
- Разрывы строк заменяет на пробелы
- Точки с запятой заменяет на разрывы строк

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Preprocessor.h строка 61

10.23.2 Конструктор(ы)

10.23.2.1 Preprocessor()

```
Preprocessor::Preprocessor (
            const std::string & fileName )
```

Конструктор, принимающий на вход имя обрабатываемого файла

После посимвольной обработки формирует промежуточный вывод intermediateOutput
Затем убирает из вывода пробелы и знаки табуляции и формирует окончательный поток вывода resultStream

Аргументы

in	fileName	константная ссылка на строку — имя обрабатываемого файла

См. определение в файле Preprocessor.cpp строка 44

```
45     :
46     inStringLastEscape(false),
47     inCharLastEscape(false),
48     inInlineLastEscape(false),
49     inMultilineLastStar(false),
50     currentParser(&Preprocessor::normalParser)
51 {
52 #pragma warning (push)
53 #pragma warning (disable: 4996)
54     FILE * inputFile = fopen(fileName.c_str(), "r");
55 #pragma warning (pop)
56
57     int symbol;
58
59     while ((symbol = fgetc(inputFile)) != EOF)
60     {
61         initialInput.push_back(symbol);
62         (this->*currentParser)(symbol, intermediateOutput);
63     }
```

```
    fclose(inputFile);

    std::stringstream ss(intermediateOutput);

    std::string readline;

    while (ss.good())
    {
        getline(ss, readline);

        readline.erase(std::remove(readline.begin(), readline.end(), ' '), readline.end());
        readline.erase(std::remove(readline.begin(), readline.end(), '\t'), readline.end());

        resultStream << readline << std::endl;
    }

    resultString = resultStream.str();
}//Preprocessor(...)
```

10.23.2.2 ~Preprocessor()

Preprocessor::~Preprocessor ( ) [inline]

Деструктор

См. определение в файле Preprocessor.h строка 161

```
161 { };
```

10.23.3 Методы

10.23.3.1 afterSlashParser()

void Preprocessor::afterSlashParser (
char ch,
std::string & str) [private]

Обработчик символа в режиме парсера выражения после слэша

При появлении символов / или * переключает парсер в режим обработки однострочного или многострочного комментария
Отальные символы выводит в поток после его предварительной обработки фукцией Preprocessor↩ ::processSymbol

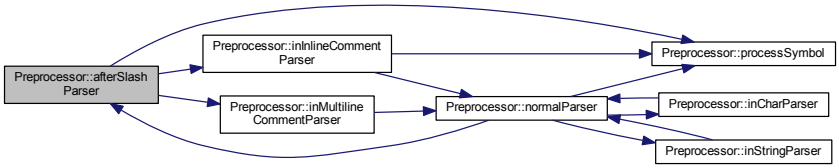
Аргументы

in	ch	обрабатываемый символ
out	str	ссылка на строку, в которую сохряняется результат

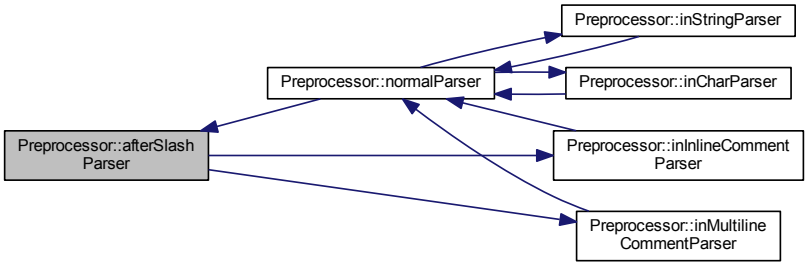
См. определение в файле Preprocessor.cpp строка 144

```
{
    switch (ch)
    {
    case '/':
        currentParser = &Preprocessor::inInlineCommentParser
;
        return;

    case '*':
        currentParser = &Preprocessor::inMultilineCommentParser
;
        return;

    default:
        str.push_back(processSymbol(ch));
    }
}//afterSlashParser(...)
```

Граф вызовов:



Граф вызова функции:



10.23.3.2 inCharParser()

void Preprocessor::inCharParser (
char ch,
std::string & str) [private]

Обработчик символа в режиме парсера символа (внутри одинарных кавычек)

При появлении парного апострофа переводит парсер в обычный режим, обрабатываемый символ в неизменном виде передается в поток

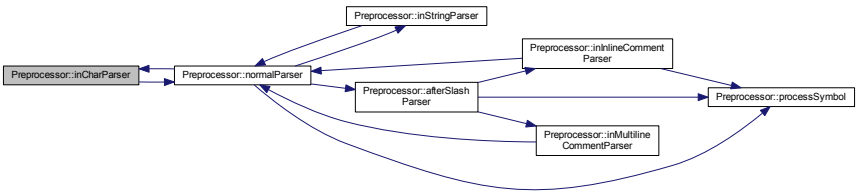
Аргументы

in	ch	обрабатываемый символ
out	str	ссылка на строку, в которую сохряняется результат

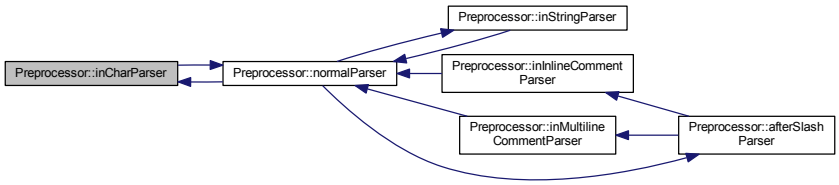
См. определение в файле Preprocessor.cpp строка 134

```
{
    str.push_back(ch);
    if (ch == '\'' && !inCharLastEscape)
        currentParser = &Preprocessor::normalParser;
    inCharLastEscape = (ch == '\\') && !inCharLastEscape;
}//inCharParser(...)
```

Граф вызовов:



Граф вызова функции:



10.23.3.3 inInlineCommentParser()

void Preprocessor::inInlineCommentParser (
char ch,
std::string & str) [private]

Обработчик символа в режиме парсера однострочного комментария (после //)

Игнорирует все симводы до появления конца строки.
При появлении конца строки переключает парсер в обычный режим и выводит конец строки в результирующую строчку

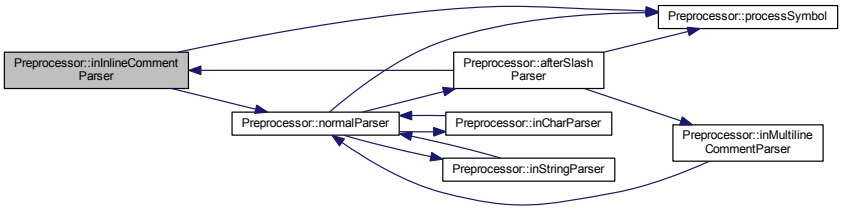
Аргументы

in	ch	обрабатываемый символ
out	str	ссылка на строку, в которую сохряняется результат

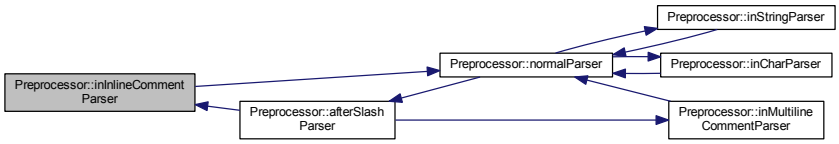
См. определение в файле Preprocessor.cpp строка 163

```
164 {
165     if (ch == '\n' && !inInlineLastEscape)
166     {
167         str.push_back(processSymbol(ch));
168         currentParser = &Preprocessor::normalParser;
169     }
170     inInlineLastEscape = (ch == '\\') && !inInlineLastEscape;
171 }//inInlineCommentParser(...)
```

Граф вызовов:



Граф вызова функции:



10.23.3.4 inMultilineCommentParser()

void Preprocessor::inMultilineCommentParser (
char ch,
std::string & str) [private]

Обработчик символа в режиме парсера многострочного комментария

Игнорирует все симводы до появления сочетания, соответствующего окончанию комментария.

При появлении символов окончания комментария выводит в результирующую строку пробел и переключает парсер в обычный режим

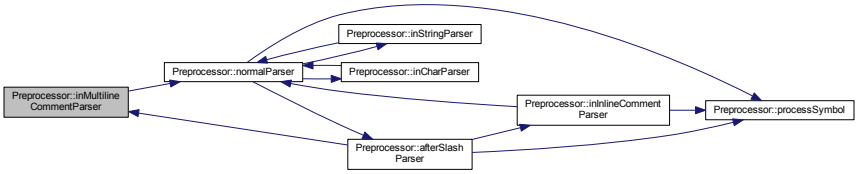
Аргументы

in	ch	обрабатываемый символ
out	str	ссылка на строку, в которую сохряняется результат

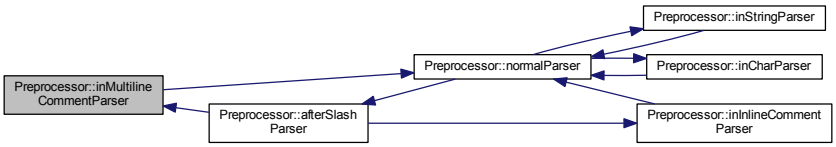
См. определение в файле Preprocessor.cpp строка 175

```
176 {
177     if (ch == '/' && inMultilineLastStar)
178     {
179         str.push_back(' ');
180         currentParser = &Preprocessor::normalParser;
181     }
182     inMultilineLastStar = (ch == '*');
183 }
```

Граф вызовов:



Граф вызова функции:



10.23.3.5 inStringParser()

void Preprocessor::inStringParser (
char ch,
std::string & str) [private]

Обработчик символа в режиме парсера строки (внутри двойных кавычек)

При появлении парной кавычки переводит парсер в обычный режим, обрабатываемый символ в неизменном виде передается в поток

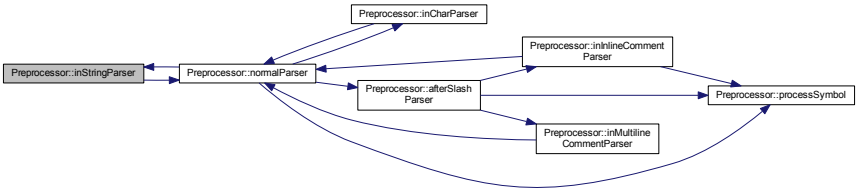
Аргументы

in	ch	обрабатываемый символ
out	str	ссылка на строку, в которую сохряняется результат

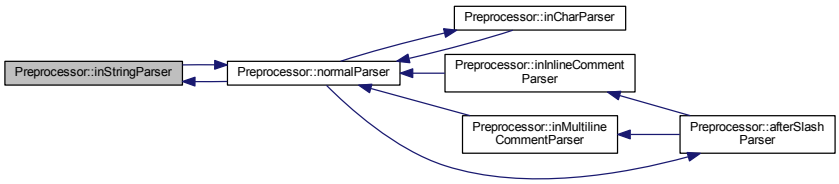
См. определение в файле Preprocessor.cpp строка 124

```
125 {
126     str.push_back(ch);
127     if (ch == '\"' && !inStringLastEscape)
128         currentParser = &Preprocessor::normalParser;
129     inStringLastEscape = (ch == '\\') && !inStringLastEscape;
130 }//inStringParser(...)
```

Граф вызовов:



Граф вызова функции:



10.23.3.6 normalParser()

void Preprocessor::normalParser (
char ch,
std::string & str) [private]

Обработчик символа в режиме обычного состояния парсера

При появлении символов ', / переключает парсер в соответствующий режим Выводит обрабатываемый символ в поток (кроме слэша, который игнорируется) после его предварительной обработки фукцией Preprocessor::processSymbol

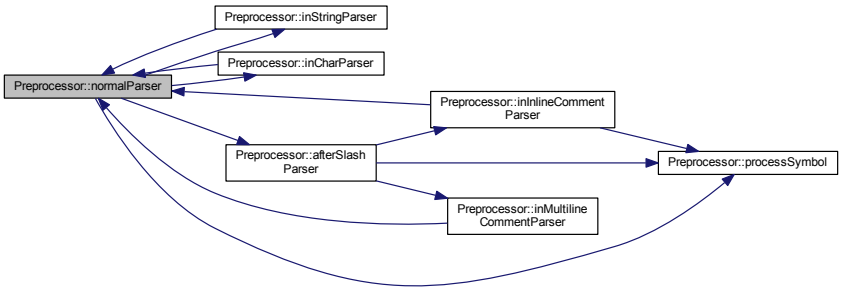
Аргументы

in	ch	обрабатываемый символ
out	str	ссылка на строку, в которую сохряняется результат

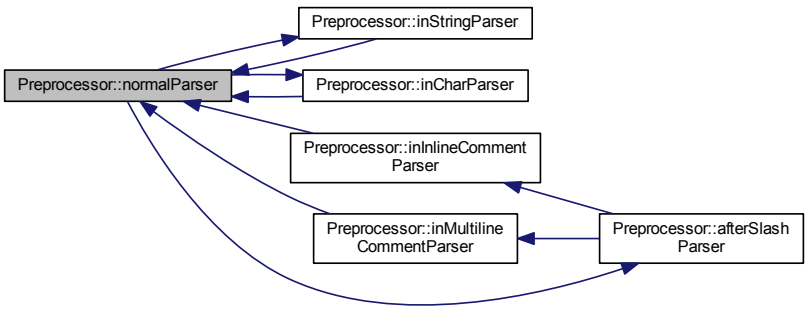
См. определение в файле Preprocessor.cpp строка 103

```
104 {
105     switch (ch)
106     {
107     case '\"':
108         currentParser = &Preprocessor::inStringParser;
109         break;
110
111     case '\'':
112         currentParser = &Preprocessor::inCharParser;
113         break;
114
115     case '/':
116         currentParser = &Preprocessor::afterSlashParser;
117         return;
118     }
119     str.push_back(processSymbol(ch));
120 }//normalParser(...)
```

Граф вызовов:



Граф вызова функции:



10.23.3.7 processSymbol()

char Preprocessor::processSymbol (
char ch) [private]

Базовая функция обработки символа В зависимости от входого символа возвращает на выход:

- (конец строки) — пробел
- ; (точка с зарятой) — конец строки
- исходный символ в остальных случаях

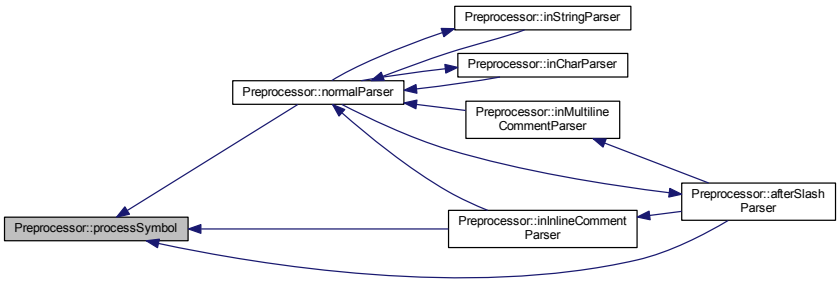
Аргументы

in	ch	обрабатываемый символ return символ, который отправляется в обработанный файл

См. определение в файле Preprocessor.cpp строка 86

```
87 {
88     switch (ch)
89     {
90     case '\n':
91         return ' ';
92
93     case ';':
94         return '\n';
95
96     default:
97         return ch;
98     }
99 }//processSymbol(...)
```

Граф вызова функции:



10.23.4 Данные класса

10.23.4.1 currentParser

void(Preprocessor::∗ Preprocessor::currentParser) (char ch, std::string &str) [private]

Опредедение currentParser как указателя на функцию-члена класса

См. определение в файле Preprocessor.h строка 77

10.23.4.2 inCharLastEscape

bool Preprocessor::inCharLastEscape [private]

Признак встречи слэша внутри символа (внутри одинарных кавычек)

См. определение в файле Preprocessor.h строка 68

10.23.4.3 inInlineLastEscape

bool Preprocessor::inInlineLastEscape [private]

Признак встречи слэша внутри однострочного комментария (после //)

См. определение в файле Preprocessor.h строка 71

10.23.4.4 initialInput

std::string Preprocessor::initialInput

Строка, содержащая исходный файл в первоначальном виде

См. определение в файле Preprocessor.h строка 145

10.23.4.5 inMultilineLastStar

bool Preprocessor::inMultilineLastStar [private]

Признак встречи звездочки внутри многострочного комментария (после '/∗')

См. определение в файле Preprocessor.h строка 74

10.23.4.6 inStringLastEscape

bool Preprocessor::inStringLastEscape [private]

Признак встречи слэша внутри строки (внутри двойных кавычек)

См. определение в файле Preprocessor.h строка 65

10.23.4.7 intermediateOutput

std::string Preprocessor::intermediateOutput

Строка, содержащая результат промежуточной обработки файла

См. определение в файле Preprocessor.h строка 148

10.23.4.8 resultStream

std::stringstream Preprocessor::resultStream

См. определение в файле Preprocessor.h строка 165

10.23.4.9 resultString

std::string Preprocessor::resultString

Строка, содержащая окончательный результат обработки файла

См. определение в файле Preprocessor.h строка 161

Объявления и описания членов классов находятся в файлах:

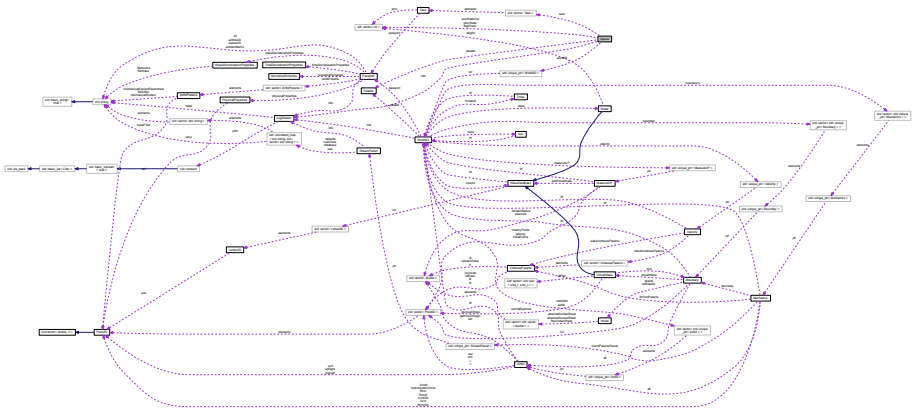
- Preprocessor/Preprocessor.h
- Preprocessor/Preprocessor.cpp

10.24 Класс Queue

Класс, опеделяющий список решаемых задач и очередь их прохождения

#include <Queue.h>

Граф связей класса Queue:



Открытые члены

- Queue (int &argc, char **&argv, void(*_CreateMpiTypes)())

  Конструктор
- ~Queue ()

  Деструктор
- void TaskSplit ()

  Процедура постановка новых задач на отсчет и занятие процессоров
- void TaskUpdate ()

  Процедура обновления состояния задач и процессоров
- void RunConveyer ()

  Запуск вычислительного конвейера (в рамках кванта времени)
- void LoadTasksList (const std::string &_tasksFile, const std::string &_mechanicsFile, const std↩ ::string &_defaultsFile, const std::string &_switchersFile)

  Загрузка списка задач

Открытые атрибуты

- std::vector< Task > task

  Список описаний решаемых задач
- std::unique_ptr< World2D > world2D

  Умный указатель на текущую решаемую задачу
- int myProcState

  Состояние данного процессора
- int myProcStateVar

  Состояние данного процессора
- const double kvantTime = 5.0

  Продолжительность кванта времени в секундах

- int currentKvant

  Номер текущего кванта времени
- int nextKvant

  Признак необходимости выполнения следующего кванта и продолжения расчета
- Parallel parallel

  Класс, опеделяющий параметры исполнения задачи в параллельном MPI-режиме
- int prepared

  Число подготовленных к запуску задач
- int solving

  Число решаемых в данный момент задач
- int finished

  Число уже решенных задач

Закрытые члены

- void ConstructProcStateVar ()

  Процедура, нумерующая задачи в возрастающем порядке
- void AddTask (int _nProc, const Passport &_passport)

  Добавление задачи в список

Закрытые данные

- struct {
  int prepared
  Число подготовленных к запуску задач
  int solving
  Число решаемых в данный момент задач
  int finished
  Число уже решенных задач
  } numberOfTask

  Структура, содержащая информацию о количестве задач в данный момент времени
- std::vector< int > procState

  Список состояний процессоров
- std::vector< int > procStateVar

  Модифицированный список состояний процессоров
- std::vector< int > flagFinish

  Список возвращаемых флагов останова задачи
- int myidAll

  Глобальный номер процессора
- int nProcAll

  Общее число процессоров
- MPI_Group groupAll

  Группа для всех процессов
- MPI_Group groupStarting

  Группа для головных процессоров в стартующих в данном кванте времени задачах
- MPI_Comm commStarting

  Коммуникатор для головных процессоров в стартующих в данном кванте времени задачах
- MPI_Group groupSolving

  Группа для головных процессоров в решаемых в данном кванте времени задачах

- MPI_Comm commSolving

  Коммуникатор для головных процессоров в решаемых в данном кванте времени задачах
- int sizeCommSolving

  Число процессоров в группе для головных процессоров в решаемых в данном кванте времени задачах
- LogStream info

  Поток для вывода логов и сообщений об ошибках

10.24.1 Подробное описание

Класс, опеделяющий список решаемых задач и очередь их прохождения

Управляет распределением задач по процессорам, инициализацией их запуска и выгрузки

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Queue.h строка 66

10.24.2 Конструктор(ы)

10.24.2.1 Queue()

```
Queue::Queue (
            int & argc,
            char **& argv,
            void(*)() _CreateMpiTypes )
```

Конструктор

Производит инициализацию очереди перед запуском всего процесса

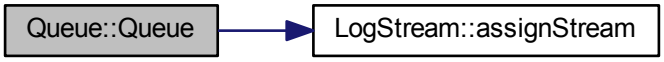
Аргументы

in	argc	ссылка на число параметров командной строки
in	argv	ссылка на указатель на список параметров командной строки
in	_CreateMpiTypes	указатель на функцию, инициализирующую все необходимые MPI-описания типов

См. определение в файле Queue.cpp строка 53

```
54 {
55     MPI_Init(&argc, &argv);
56     MPI_Comm_size(MPI_COMM_WORLD, &nProcAll);
57     MPI_Comm_rank(MPI_COMM_WORLD, &myidAll);
58     MPI_Comm_group(MPI_COMM_WORLD, &groupAll);
59     _CreateMpiTypes();
60
61     //готовим систему к запуску
62     //(выполняется на глобальном нулевом процессоре)
63     if (myidAll == 0)
64     {
65         info.assignStream(defaults::defaultQueueLogStream, "
    queue");
66
67         //Устанавливаем флаг занятости в состояние "свободен" всем процессорам,
68         procState.resize(nProcAll, MPI_UNDEFINED);
69         procStateVar.resize(nProcAll, MPI_UNDEFINED);
70
71         numberOfTask.solving  = 0; //число решаемых в данный момент задач
72         numberOfTask.prepared = 0; //число подготовленных к запуску задач (в том числе)
73         numberOfTask.finished = 0; //число уже отсчитанных задач
74
75         //текущий номер кванта времени
76         currentKvant = -1;
77     }//if myid==0
78 }//Queue()
```

Граф вызовов:



10.24.2.2 ~Queue()

Queue::~Queue ( )

Деструктор

См. определение в файле Queue.cpp строка 82

```
83 {
84     info('i') << "Goodbye!" << std::endl;
85     MPI_Finalize();
86 }//~Queue()
```

10.24.3 Методы

10.24.3.1 AddTask()

void Queue::AddTask (
int _nProc,
const Passport & _passport) [private]

Добавление задачи в список

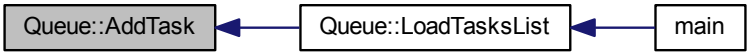
Аргументы

in	_nProc	число запрашиваемых процессоров
in	_passport	константная ссылка на паспорт задачи, направляемой в очередь

См. определение в файле Queue.cpp строка 517

```
518 {
519     Task newTask(_passport);
520     newTask.nProc = _nProc;
521     newTask.state = TaskState::waiting;
522     newTask.proc.resize(_nProc);
523
524     task.emplace_back(newTask);
525 }//AddTask(...)
```

Граф вызова функции:



10.24.3.2 ConstructProcStateVar()

void Queue::ConstructProcStateVar ( ) [private]

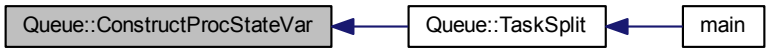
Процедура, нумерующая задачи в возрастающем порядке

Необходима для корректного распределения задач по процессорам

См. определение в файле Queue.cpp строка 459

```
460 {
461     std::vector<bool> prFlag;
462     int number = -1;
463
464     prFlag.resize(nProcAll, true); //их надо просматривать
465
466     for (int i = 0; i<nProcAll; i++)
467     if (procState[i] == MPI_UNDEFINED)
468     {
469         prFlag[i] = false;  //уже просмотрели
470         procStateVar[i] = MPI_UNDEFINED;
471     } //if (ProcState[i]==MPI_UNDEFINED)
472
473     for (int i = 0; i<nProcAll; i++)
474     if (prFlag[i])
475     {
476         prFlag[i] = false;
477         number++;
478
479         for (int s = i; s<nProcAll; s++)
480         if (procState[s] == procState[i])
481         {
482             procStateVar[s] = number;
483             prFlag[s] = false;
484         }//if (ProcState[s]==ProcState[i])
485     } //if (pr_flag[i])
486 }//ConstructProcStateVar()
```

Граф вызова функции:



10.24.3.3 LoadTasksList()

```
void Queue::LoadTasksList (
            const std::string & _tasksFile,
            const std::string & _mechanicsFile,
            const std::string & _defaultsFile,
            const std::string & _switchersFile )
```

Загрузка списка задач

Аргументы

in	_tasksFile	константная ссылка на имя файла с описанием очереди задач
in	_mechanicsFile	константная ссылка на имя файла со словарем механических систем
in	_defaultsFile	константная ссылка на имя файла с описанием параметров по умолчанию
in	_switchersFile	константная ссылка на имя файла со значениями параметров-переключателей

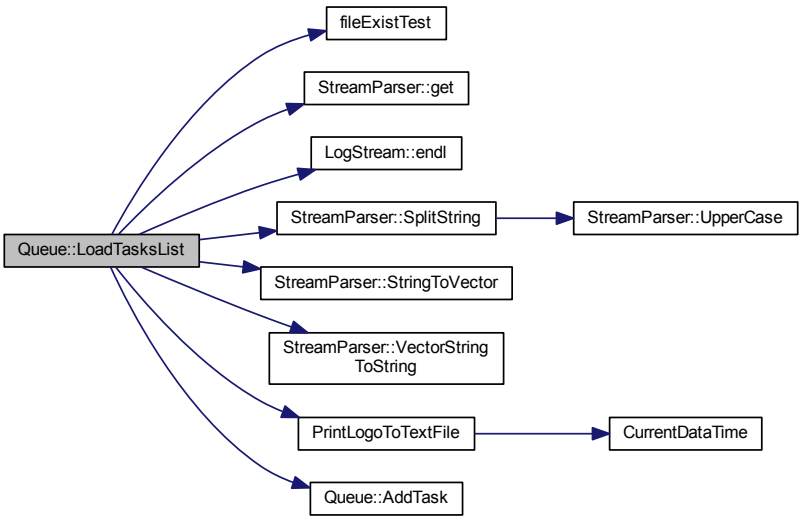
См. определение в файле Queue.cpp строка 529

```
530 {
531     if (
532     fileExistTest(_tasksFile, info) &&
533     fileExistTest(_defaultsFile, info) &&
534     fileExistTest(_switchersFile, info)
535     )
536     {
537         std::stringstream tasksFile(Preprocessor(_tasksFile).resultString);
538         std::stringstream defaultsFile(Preprocessor(_defaultsFile).resultString);
539         std::stringstream switchersFile(Preprocessor(_switchersFile).resultString);
540         std::vector<std::string> taskFolder;
541
542         std::unique_ptr<StreamParser> parserTaskList;
543         parserTaskList.reset(new StreamParser(info, "parser", tasksFile, '(', ')'));
544
545         std::vector<std::string> alltasks;
546         parserTaskList->get("problems", alltasks);
547
548         std::ptrdiff_t nTasks = std::count_if(alltasks.begin(), alltasks.end(), [](const std::string& a) {
    return (a.length() > 0);});
549         info.endl();
550         info('i') << "Number of problems to be solved: " << nTasks << std::endl;
551
552         for (size_t i = 0; i < alltasks.size(); ++i)
553         {
554             if (alltasks[i].length() > 0)
555             {
556                 //делим имя задачи + выражение в скобках на 2 подстроки
557                 std::pair<std::string, std::string> taskLine =
```

```
StreamParser::SplitString(info, alltasks[i], false);

            std::string dir = taskLine.first;

            info.endl();
            info('i') << "-------- Loading problem #" << i << " (" << dir << ") --------" << std::endl;

            //вторую подстроку разделяем на вектор из строк по запятым, стоящим вне фигурных скобок
            std::vector<std::string> vecTaskLineSecond = StreamParser::StringToVector(taskLine.second, '{', '}');

            //создаем парсер и связываем его с параметрами профиля
            std::stringstream tskStream(StreamParser::VectorStringToString(vecTaskLineSecond));
            std::unique_ptr<StreamParser> parserTask;

            parserTask.reset(new StreamParser(info, "problem parameters", tskStream, defaultsFile, {"pspfile", "np", "copyPspFile"}));

            std::string pspFile;
            int np;
            //считываем нужные параметры с учетом default-значений
            parserTask->get("pspfile", pspFile, &defaults::defaultPspFile);
            parserTask->get("np", np, &defaults::defaultNp);

            if (np > nProcAll)
            {
                info.endl();
                info('e') << "problem #" << i << " (" << dir << \
                    ") requires " << np << " processors (" << nProcAll << " available)" << std::endl;
                exit(-1);
            }
            else
            {
                info('i') << "number of processors np = " << np << std::endl;
            }

            std::string copyPspFile;
            parserTask->get("copyPspFile", copyPspFile, &defaults::defaultCopyPspFile);
            if (copyPspFile.length() > 0)
            {
                //Копировать паспорт поручаем только одному процессу
                if (myidAll == 0)
                {
#if defined(_WIN32)
                    _mkdir(dir.c_str());
#else
                    mkdir(dir.c_str(), S_IRWXU | S_IRGRP | S_IROTH);
#endif

                    std::ofstream outPassportFile;
                    outPassportFile.open("./" + dir + "/" + pspFile);
                    PrintLogoToTextFile(outPassportFile, "./" + dir + "/" + pspFile, "Copy of the existing passport ./" + copyPspFile);

                    std::stringstream ss(Preprocessor(copyPspFile).resultString);

                    std::string readline;

                    while (ss.good())
                    {
                        getline(ss, readline);

                        readline.erase(remove(readline.begin(), readline.end(), ' '), readline.end());
                        readline.erase(remove(readline.begin(), readline.end(), '\t'), readline.end());

                        if (readline.length() > 0)
                        {
                            outPassportFile << readline << ";" << std::endl;
                            outPassportFile.flush();
                        }
                    }

                    outPassportFile.close();
                }
            }

            MPI_Barrier(MPI_COMM_WORLD);


            Passport psp(info, dir, i, pspFile, _mechanicsFile, _defaultsFile, _switchersFile, vecTaskLineSecond);
            AddTask(np, psp);

```

```
            info('i') << "-------- Problem #" << i << " (" << dir << ") is loaded --------" << std::endl << std::endl;
        }
    } //for i

    //tasksFile.close();
    tasksFile.clear();

    //defaultsFile.close();
    defaultsFile.clear();

    //switchersFile.close();
    switchersFile.clear();

  }
  else
  {
    exit(1);
  }

}
```

Граф вызовов:



Граф вызова функции:

10.24.3.4 RunConveyer()

void Queue::RunConveyer ( )

Запуск вычислительного конвейера (в рамках кванта времени)

См. определение в файле Queue.cpp строка 490

```
491 {
492     if (parallel.commWork != MPI_COMM_NULL)
493     {
494         double kvantStartWallTime; //Время на главном процессоре, когда начался очередной квант
495         double deltaWallTime;
496
497         if (parallel.myidWork == 0)
498             kvantStartWallTime = MPI_Wtime();
499
500         do
501         {
502             if (!world2D->isFinished())
503                 world2D->Step();
504
505             if (parallel.myidWork == 0)
506                 deltaWallTime = MPI_Wtime() - kvantStartWallTime;
507             MPI_Bcast(&deltaWallTime, 1, MPI_DOUBLE, 0, parallel.
    commWork);
508         }
509
510         //Проверка окончания кванта по времени (или завершения задачи)
511         while (deltaWallTime < kvantTime);
512     }//if (commWork != MPI_COMM_NULL)
513 }//RunConveyer()
```

Граф вызова функции:

10.24.3.5 TaskSplit()

void Queue::TaskSplit ( )

Процедура постановка новых задач на отсчет и занятие процессоров

Выполняется в начале очередного кванта

См. определение в файле Queue.cpp строка 90

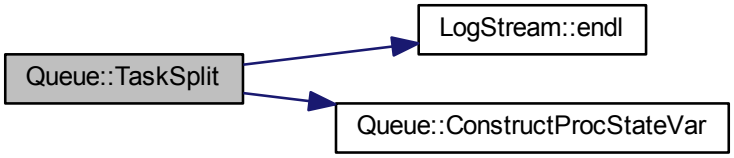
```
{
    //Пересылаем информацию о состоянии процессоров на все компьютеры
    MPI_Scatter(procState.data(), 1, MPI_INT, &myProcState, 1, MPI_INT, 0, MPI_COMM_WORLD);

    if (myProcState == MPI_UNDEFINED)
        world2D.reset(nullptr);

    if (myidAll == 0)
    {
        currentKvant++;

        int nfree = 0; //число свободных процессоров

        //Обнуляем число готовых к старту задач
        numberOfTask.prepared = 0;

        //считаем число свободных процессоров
        for (int i = 0; i < nProcAll; i++)
        if (procState[i] == MPI_UNDEFINED)
            nfree++;

        //формируем номер следующей задачи, которая возможно будет поставлена на отсчет:
        size_t taskFol = numberOfTask.finished + numberOfTask.solving;

        //проверяем, достаточно ли свободных процессоров для запуска еще одной задачи
        //при условии, что не все задачи уже решены
        while ((taskFol < task.size()) && (nfree >= task[taskFol].nProc))
        {
            int p = 0; //число найденных свободных процессоров под задачу
            int j = 0; //текущий счетчик процессоров

            //подбираем свободные процессоры
            do
            {
                if (procState[j] == MPI_UNDEFINED)
                {
                    //состояние процессора устанавливаем в "занят текущей задачей"
                    procState[j] = static_cast<int>(taskFol);

                    //номер процессора посылаем в перечень процессоров, решающих текущую задачу
                    task[taskFol].proc[p] = j;

                    //увеличиваем число найденных процессоров на единицу
                    p++;
                }//if (ProcState[j]==MPI_UNDEFINED)

                //переходим к следующему процессору
                j++;
            } while (p < task[taskFol].nProc);

            //состояние задачи устанавливаем в режим "стартует"
            task[taskFol].state = TaskState::starting;

            //отмечаем номер кванта, когда задача начала считаться
            task[taskFol].startEndKvant.first = currentKvant;

            //изменяем счетчики числа задач
            numberOfTask.prepared++;
            numberOfTask.solving++;

            //изменяем счетчик свободных процессоров
            nfree -= task[taskFol].nProc;

            //переходим к следующей задаче
            taskFol++;

        }//while ((nfree >= Task[task_fol].nproc)&&(task_fol<Task.size()))

    }//if (myidAll == 0)

    //Вывод информации о занятости процессоров задачами
    if (myidAll == 0)
    {
        info.endl();
        info('i') << "ProcStates: " << std::endl;
        for (int i = 0; i < nProcAll; i++)
            info('-') << "proc[" << i << "] <=> problem[" << procState[i] << "]" << std::endl;
        info.endl();
    }

    //Пересылаем информацию о состоянии процессоров на все компьютеры
    MPI_Scatter(procState.data(), 1, MPI_INT, &myProcState, 1, MPI_INT, 0, MPI_COMM_WORLD);

    //Синхронизируем информацию
    MPI_Bcast(&numberOfTask.solving, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

```
    MPI_Bcast(&numberOfTask.prepared, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&numberOfTask.finished, 1, MPI_INT, 0, MPI_COMM_WORLD);

    //список головных процессоров на тех задачах,
    //которые только что поставлены на счет:
    std::vector<int> prepList;

    //список паспортов для стартующих задач
    //vector<Passport> pspList;

    //формируем списки prepList и pspList
    //(выполняется на глобальном нулевом процессоре)
    if (myidAll == 0)
    {
        for (size_t i = 0; i<task.size(); ++i)
        //находим задачи в состоянии "стартует"
        if (task[i].state == TaskState::starting)
        {
            //запоминаем номер того процессора, что является там головным
            prepList.push_back(task[i].proc[0]);

            //состояние задачи вереводим в "считает"
            task[i].state = TaskState::running;
        }
    }//if myidAll==0


    //Рассылаем количество стартующих задач в данном кванте на все машины
    MPI_Bcast(&numberOfTask.prepared, 1, MPI_INT, 0, MPI_COMM_WORLD);

    if (myidAll > 0)
    {
        prepList.resize(numberOfTask.prepared);
    }//if myid==0

    //следующий фрагмент выполняется только если в данном кванте стартуют новые задачи
    if (numberOfTask.prepared > 0)
    {
        //пересылаем список головных машин стартующих задач на все процессоры
        MPI_Bcast(prepList.data(), numberOfTask.prepared, MPI_INT, 0, MPI_COMM_WORLD);

        //формируем группу и коммуникатор стартующих головных процессоров
        commStarting = MPI_COMM_NULL;
        MPI_Group_incl(groupAll, numberOfTask.prepared, prepList.data(), &groupStarting);
        MPI_Comm_create(MPI_COMM_WORLD, groupStarting, &commStarting);


        //следующий фрагмент кода только для стартующих процессов,
        //которые входят в коммуникатор comm_starting
        if (commStarting != MPI_COMM_NULL)
        {

            //получаем номер процесса в списке стартующих
            int myidStarting;
            MPI_Comm_rank(commStarting, &myidStarting);

            //подготовка стартующих задач
            parallel.myidWork = 0;
            world2D.reset(new World2D(task[myProcState].passport, parallel));

            //Коммуникатор головных процессоров стартующих задач
            //выполнил свое черное дело и будет удален
            MPI_Comm_free(&commStarting);
        } //if(comm_starting != MPI_COMM_NULL)

        //их группа тоже больше без надобности
        MPI_Group_free(&groupStarting);
    }//if (task_prepared>0)

    //формируем группу и коммуникатор считающих головных процессоров
    //в том числе тех, которые стартуют, и тех, которые ранее входили в commStarting
    std::vector<int> solvList; //список номеров головных процессов

    if (myidAll == 0)
    {
        //Если нулевой просессор свободен
        //(не является головным в решении задачи в данном кванте) -
        //- формально присоединяем его в группу головных
        //необходимо для корректного обмена данными
        if (procState[0] == MPI_UNDEFINED)
            solvList.push_back(0);

        //далее ищем считающие задачи (в том числе только что стартующие)
        //и собираем номера их головных процессоров
        //(собираем в порядке просмотра номеров процессоров))
```

```
        for (int s = 0; s<nProcAll; s++)
        if ((procState[s] != MPI_UNDEFINED) && (task[procState[s]].state == TaskState::running) && (task[procState[s]].proc[0] == s))
            solvList.push_back(s);

        sizeCommSolving = static_cast<int>(solvList.size());

        flagFinish.clear();
        flagFinish.resize(solvList.size());

    }//if myidAll==0

    //число sizeCommSolving на единицу больше, чем taskSolving, если нулевой процессор свободен
    //и равно taskSolving, если он занят решением задачи
    //(если он занят решением задачи, то непременно является головным)

    //пересылаем число головных процессоров и их список на все компьютеры
    //(в том числе головной процессор - независимо от его состояния)
    MPI_Bcast(&sizeCommSolving, 1, MPI_INT, 0, MPI_COMM_WORLD);

    if (myidAll > 0)
        solvList.resize(sizeCommSolving);

    MPI_Bcast(solvList.data(), sizeCommSolving, MPI_INT, 0, MPI_COMM_WORLD);

    commSolving = MPI_COMM_NULL;

    //формируем группу и коммуникатор головных процессоров
    MPI_Group_incl(groupAll, sizeCommSolving, solvList.data(), &groupSolving);
    MPI_Comm_create(MPI_COMM_WORLD, groupSolving, &commSolving);

    if (myidAll == 0)
    {
        //составление неубывающего массива номеров для верного
        //распределения задач по процессоров, с тем чтобы задачи
        //с прошлого кванта оказались в тех же группах, что и раньше
        ConstructProcStateVar();
    }

    //Пересылаем информацию о состоянии процессоров на все компьютеры
    MPI_Scatter(procStateVar.data(), 1, MPI_INT, &myProcStateVar, 1, MPI_INT, 0, MPI_COMM_WORLD);

    //Расщепляем весь коммуникатор MPI_COMM_WORLD (т.е. вообще все процессоры)
    //на коммуникаторы решаемых задач
    //В результате все процессоры, решающие конкретную задачу,
    //объединяются в коммуникатор comm_work
    //Несмотря на то, что он называется везде одинаково, у каждой задачи он свой
    //и объединяет ровно те процессоры, которые надо
    //Процессоры, состояние которых установлено в "свободен", т.е.
    //ProcState=MPI_UNDEFINED (=-32766) ни в один
    //новый коммуникатор commWork не попадут
    MPI_Comm_split(MPI_COMM_WORLD, myProcStateVar, 0, &parallel.commWork);

    //Рассылаем величину кванта времени
    //MPI_Bcast(&kvantTime, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    //следующий код выполняется процессорами, участвующими в решении задач,
    //следовательно, входящими в коммуникаторы comm_work
    if (parallel.commWork != MPI_COMM_NULL)
    {
        //оперделяем "локальный" номер данного процессора в коммуникаторе commWork и число процессов в нем
        MPI_Comm_size(parallel.commWork, &parallel.nProcWork);
        MPI_Comm_rank(parallel.commWork, &parallel.myidWork);

        //World2D равен nullptr только в случае, когда выполняется первый шаг
        //в этом случае он создан только на главном процессоре группы;
        //на остальных происходит его создание
        if (world2D == nullptr)
            world2D.reset(new World2D(task[myProcState].passport, parallel));

        //Синхронизация параметров
        MPI_Bcast(&(world2D->getPassport().physicalProperties.currTime), 1, MPI_DOUBLE, 0, parallel.commWork);

        //Создание файлов для записи сил
        if ((parallel.myidWork == 0) && (world2D->getCurrentStep() == 0))
        for (size_t q = 0; q < world2D->getPassport().airfoilParams.size(); ++q)
            world2D->GenerateMechanicsHeader(q);

        //Создание файла для записи временной статистики
        if ((parallel.myidWork == 0) && (world2D->getCurrentStep() == 0))
            world2D->getTimestat().GenerateStatHeader();
```

```
341
342         //TODO
343         //Синхронизация паспортов
344         //MPI_Bcast(...)
345     }
346
347 }//TaskSplit()
```

Граф вызовов:



Граф вызова функции:

10.24.3.6 TaskUpdate()

void Queue::TaskUpdate ( )

Процедура обновления состояния задач и процессоров

Выполняется в конце очередного кванта

См. определение в файле Queue.cpp строка 351

```
352 {
353     info('i') << "------ Kvant finished ------" << std::endl;
354
355     //Код только для головных процессов, которые входят в коммуникатор commSolving
356     //т.е. выполняется только на головных процессорах, решающих задачи.
357     //Кажется, что можно было бы с тем же эфектом написать здесь if (myidWork==0),
358     //но это не так, поскольку нужно включить в себя еще нулевой процессор,
359     //который в любом случае присоединен к коммуникатору comm_solving,
360     //даже если он не решает задачу (а если решает - то он всегда головной)
361     if (commSolving != MPI_COMM_NULL)
362     {
363         //определяем номер процессора в данном коммуникаторе -
```

```
        // - вдруг потребуется на будущее!
        int myidSolving;
        MPI_Comm_rank(commSolving, &myidSolving);

        //Алгоритм возвращения результатов расчета конкретной задачи
        //Если срабатывает признак того, что решение задачи можно прекращать
        //или не решается ни одна задача (а в комуникатор входит
        //только 0-й процессор, присоединенний туда насильно)
        int stopSignal = ( (parallel.commWork == MPI_COMM_NULL) || (world2D->isFinished())) ? 1 : 0;

        //пересылаем признаки прекращения решения задач на нулевой процессор
        //коммуникатора comm_solving - т.е. на глобальный процессор с номером 0 ---
        // --- вот для чего его насильно присоединяли к этому коммуникатору
        //если сам по себе он туда не входил
        MPI_Gather(&stopSignal, 1, MPI_INT, flagFinish.data(), 1, MPI_INT, 0, commSolving);

        //после пересылки информации коммуникатор comm_solving уничтожается
        MPI_Comm_free(&commSolving);

    } //if(comm_solving != MPI_COMM_NULL)

    //и соотвествующая ему группа тоже удаляется
    MPI_Group_free(&groupSolving);


    if (parallel.commWork != MPI_COMM_NULL)
    {
        //К этому месту все процессоры, решающие задачи, приходят уже выполнив все расчеты
        //в рамках одного кванта времени
        //Поэтому коммуникаторы решаемых задач нам уже ни к чему - уничтожаем их
        MPI_Comm_free(&parallel.commWork);
    }

    //Обновление состояния задач и высвобождение процессоров из отсчитавших задач
    //(выполняется на глобальном нулевом процессоре)
    if (myidAll == 0)
    {
        //Список номеров задач, которые сейчас считаются
        std::vector<int> taskList;

        //если состояние "считается" запоминаем эту задачу
        //задачи запоминаются в порядке просмотра процессоров
        for (int s = 0; s<nProcAll; s++)
        if ( (procState[s] != MPI_UNDEFINED) && (task[procState[s]].state == TaskState::running) && (task[procState[s]].proc[0] == s) )
            taskList.push_back(procState[s]);

        //Тем задачам, которые во флаге прекращения счета вернули "1",
        //ставим состояние завершения счета.
        //Конструкция +sizeCommSolving-taskSolving введена для смещения в массиве на единицу
        //если нулевой процессор свободен - тогда он присоединяется формально в нулевом
        //элементе массива flagFinish, а содержательная часть массива смещается на 1
        //(в этом случае sizeCommSolving как раз будет на 1 больше, чем taskSolving, см. выше)
        //если же нулевой процесс занят решением задачи, то смещения нет,
        //поскольку в этом случае sizeCommSolving и taskSolving равны
        for (int i = 0; i < numberOfTask.solving; i++)
        if (flagFinish[i + sizeCommSolving - numberOfTask.solving] == 1)
            task[taskList[i]].state = TaskState::finishing;
    } //if myid==0

    if (myidAll == 0)
    {
        for (size_t i = 0; i < task.size(); i++)
        {
            //освобождаем процессоры от сосчитавшихся задач
            if (task[i].state == TaskState::finishing)
            {
                //устанавливаем состояние задачи в "отсчитано"
                task[i].state = TaskState::done;

                //отмечаем последний квант, когда задача считалась
                task[i].startEndKvant.second = currentKvant;

                //изменяем счетчики
                numberOfTask.solving--;
                numberOfTask.finished++;

                //состояние процессоров, решавших данную задачу, устанавливаем в "свободен"
                for (int p = 0; p < task[i].nProc; ++p)
                {
                    procState[task[i].proc[p]] = MPI_UNDEFINED;
                }//for p
            }//if (Task[i].state == 3)
        }//for i
```

```
447
448         //определяем, надо ли еще выделять квант времени или можно заканчивать работу
449         //путем сравнения числа отсчитанных задач с общим числом задач
450         nextKvant = (numberOfTask.finished < static_cast<int>(
    task.size())) ? 1 : 0;
451     }//if (myid == 0)
452
453
454     MPI_Bcast(&nextKvant, 1, MPI_INT, 0, MPI_COMM_WORLD);
455 }//TaskUpdate()
```

Граф вызова функции:

10.24.4 Данные класса

10.24.4.1 commSolving

MPI_Comm Queue::commSolving [private]

Коммуникатор для головных процессоров в решаемых в данном кванте времени задачах

К нему всегда присоединяется 0-й процессор, даже если он не раешает задачу в данном кванте времени

См. определение в файле Queue.h строка 144

10.24.4.2 commStarting

MPI_Comm Queue::commStarting [private]

Коммуникатор для головных процессоров в стартующих в данном кванте времени задачах

См. определение в файле Queue.h строка 134

10.24.4.3 currentKvant

int Queue::currentKvant

Номер текущего кванта времени

См. определение в файле Queue.h строка 196

10.24.4.4 finished

int Queue::finished

Число уже решенных задач

См. определение в файле Queue.h строка 89

10.24.4.5 flagFinish

std::vector<int> Queue::flagFinish [private]

Список возвращаемых флагов останова задачи

Позиции 0, 1, 2 и т.д. заполняется головными процессами, решающими задачи

Предупреждения

> Нулевая позиция — всегда соответствует процессору с myid=0 (глобальному) даже если он не участвует в данном кванте в решении задачи; в этом случае головные процессы заполняют позиции 1, 2, 3 и т.д.

См. определение в файле Queue.h строка 119

10.24.4.6 groupAll

MPI_Group Queue::groupAll [private]

Группа для всех процессов

См. определение в файле Queue.h строка 128

10.24.4.7 groupSolving

MPI_Group Queue::groupSolving [private]

Группа для головных процессоров в решаемых в данном кванте времени задачах

К ней всегда присоединяется 0-й процессор, даже если он не раешает задачу в данном кванте времени

См. определение в файле Queue.h строка 139

10.24.4.8 groupStarting

MPI_Group Queue::groupStarting [private]

Группа для головных процессоров в стартующих в данном кванте времени задачах

См. определение в файле Queue.h строка 131

10.24.4.9 info

LogStream Queue::info [private]

Поток для вывода логов и сообщений об ошибках

См. определение в файле Queue.h строка 163

10.24.4.10 kvantTime

const double Queue::kvantTime = 5.0

Продолжительность кванта времени в секундах

В пределах кванта по времени не производится глобальной синхронизации всех процессоров, решающих все задачи. Проверка состояния очереди и ее обновление производятся между квантами. По исчерпании кванта времени выполнение шагов по времени всеми процессорами прекращается до обновления очереди

См. определение в файле Queue.h строка 193

10.24.4.11 myidAll

int Queue::myidAll [private]

Глобальный номер процессора

См. определение в файле Queue.h строка 122

10.24.4.12 myProcState

int Queue::myProcState

Состояние данного процессора

Принимает значения :

- MPI_UNDEFINED (=-32766, но в зависимости от реализации MPI может быть и другой константой) — свободен;
- xxx — занят задачей номер xxx (номер из списка задач).

См. определение в файле Queue.h строка 179

10.24.4.13 myProcStateVar

int Queue::myProcStateVar

Состояние данного процессора

Принимает значения :

- MPI_UNDEFINED (=-32766, но в зависимости от реализации MPI может быть и другой константой) — свободен;
- xxx — занят задачей номер xxx (номер из списка задач).

См. определение в файле Queue.h строка 186

10.24.4.14 nextKvant

int Queue::nextKvant

Признак необходимости выполнения следующего кванта и продолжения расчета

Принимает значения:

- 1 — делать еще один квант (еще есть что считать),
- 0 — расчет окончен, выход (все расчеты выполнены).

См. определение в файле Queue.h строка 203

10.24.4.15 nProcAll

int Queue::nProcAll [private]

Общее число процессоров

См. определение в файле Queue.h строка 125

10.24.4.16 numberOfTask

struct { ... } Queue::numberOfTask [private]

Структура, содержащая информацию о количестве задач в данный момент времени

Содержит следующие поля:

- prepared — число подготовленных к запуску задач;
- solving — число решаемых в данный момент задач;
- finished — число уже решенных задач.

10.24.4.17 parallel

Parallel Queue::parallel

Класс, опеделяющий параметры исполнения задачи в параллельном MPI-режиме

Внутри него содержатся такие параметры, как:

- commWork — коммуникатор для решения конкретной задачи;
- myidWork — локальный номер данного процессора в коммуникаторе процессоров, решающих конкретную задачу;
- nProcWork — число процессоров, решающих конкретную задачу.

См. определение в файле Queue.h строка 211

10.24.4.18 prepared

int Queue::prepared

Число подготовленных к запуску задач

Включено в число решаемых в данный момент задач

См. определение в файле Queue.h строка 81

10.24.4.19 procState

std::vector<int> Queue::procState [private]

Список состояний процессоров

Длина списка равна числу процессоров
Заполняется только для на главном узле, у которого myid=0
Принимает значения:

- MPI_UNDEFINED (=-32766, но в зависимости от реализации MPI может быть и другой константой) — свободен;
- xxx — занят задачей номер xxx (номер из списка задач).

См. определение в файле Queue.h строка 99

10.24.4.20 procStateVar

std::vector<int> Queue::procStateVar [private]

Модифицированный список состояний процессоров

Длина списка равна числу процессоров
Заполняется только на главном узле, у которого myid=0
Принимает значения:

- MPI_UNDEFINED (=-32766, но в зависимости от реализации MPI может быть и другой константой) — свободен;
- xxx — занят задачей номер xxx (номер из списка задач).
  Аналогичен procState, но отличается от него следующим:
- процессы, занятые одной задачей, получают один и тот же номер;
- позиции, в которых впервые появляются номера в этом списке, образуют возрастающую последовательность.

См. определение в файле Queue.h строка 111

10.24.4.21 sizeCommSolving

int Queue::sizeCommSolving [private]

Число процессоров в группе для головных процессоров в решаемых в данном кванте времени задачах

В него всегда входит 0-й процессор, даже если он не раешает задачу в данном кванте времени

См. определение в файле Queue.h строка 149

10.24.4.22 solving

int Queue::solving

Число решаемых в данный момент задач

Включает число подготовленных к запуску задач

См. определение в файле Queue.h строка 86

10.24.4.23 task

std::vector<Task> Queue::task

Список описаний решаемых задач

В описаниях содержится информация о прохождении задач и их текущем состоянии

См. определение в файле Queue.h строка 169

10.24.4.24 world2D

std::unique_ptr<World2D> Queue::world2D

Умный указатель на текущую решаемую задачу

См. определение в файле Queue.h строка 172

Объявления и описания членов классов находятся в файлах:

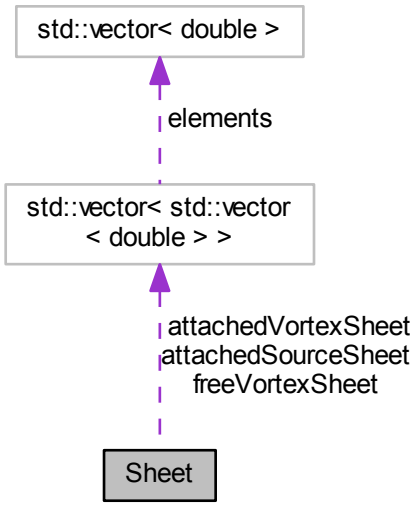
- Queue/Queue.h
- Queue/Queue.cpp

10.25 Класс Sheet

Класс, опеделяющий слои на поверхности обтекаемого профиля

#include <Sheet.h>

Граф связей класса Sheet:



Открытые члены

- Sheet ()

  Пустой конструктор
- ~Sheet ()

  Деструктор
- void SetLayersDim (size_t np, size_t layerDim)

  Установка размерностей всех векторов и их обнуление

Открытые атрибуты

- std::vector< std::vector< double > > freeVortexSheet

  Список из характеристик свободного вихревого слоя на панелях
- std::vector< std::vector< double > > attachedVortexSheet

  Список из характеристик присоединенного вихревого слоя на панелях
- std::vector< std::vector< double > > attachedSourceSheet

  Список из характеристик присоединенного слоя источников на панелях

10.25.1 Подробное описание

Класс, опеделяющий слои на поверхности обтекаемого профиля

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Sheet.h строка 54

10.25.2 Конструктор(ы)

10.25.2.1 Sheet()

Sheet::Sheet ( ) [inline]

Пустой конструктор

См. определение в файле Sheet.h строка 68

```
68 { };
```

10.25.2.2 ~Sheet()

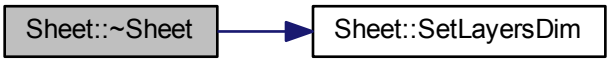
Sheet::~Sheet ( ) [inline]

Деструктор

См. определение в файле Sheet.h строка 71

```
71 { };
```

Граф вызовов:



10.25.3 Методы

10.25.3.1 SetLayersDim()

```
void Sheet::SetLayersDim (
            size_t np,
            size_t layerDim )
```

Установка размерностей всех векторов и их обнуление

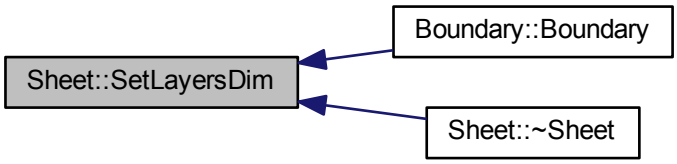
Аргументы

in	np	число панелей на профиле (внешняя размерность списков)
in	layerDim	количество чисел, которыми характеризуются слои на каждой из панелей

См. определение в файле Sheet.cpp строка 44

```
45 {
46     freeVortexSheet.resize(np);
47     attachedVortexSheet.resize(np);
48     attachedSourceSheet.resize(np);
49
50     for (size_t j = 0; j < np; ++j)
51     {
52         freeVortexSheet[j].resize(layerDim, 0.0);
53         attachedVortexSheet[j].resize(layerDim, 0.0);
54         attachedSourceSheet[j].resize(layerDim, 0.0);
55     }
56 }
```

Граф вызова функции:



10.25.4 Данные класса

10.25.4.1 attachedSourceSheet

std::vector<std::vector<double> > Sheet::attachedSourceSheet

Список из характеристик присоединенного слоя источников на панелях

См. определение в файле Sheet.h строка 65

10.25.4.2 attachedVortexSheet

std::vector<std::vector<double> > Sheet::attachedVortexSheet

Список из характеристик присоединенного вихревого слоя на панелях

См. определение в файле Sheet.h строка 62

10.25.4.3 freeVortexSheet

std::vector<std::vector<double> > Sheet::freeVortexSheet

Список из характеристик свободного вихревого слоя на панелях

См. определение в файле Sheet.h строка 59

Объявления и описания членов классов находятся в файлах:

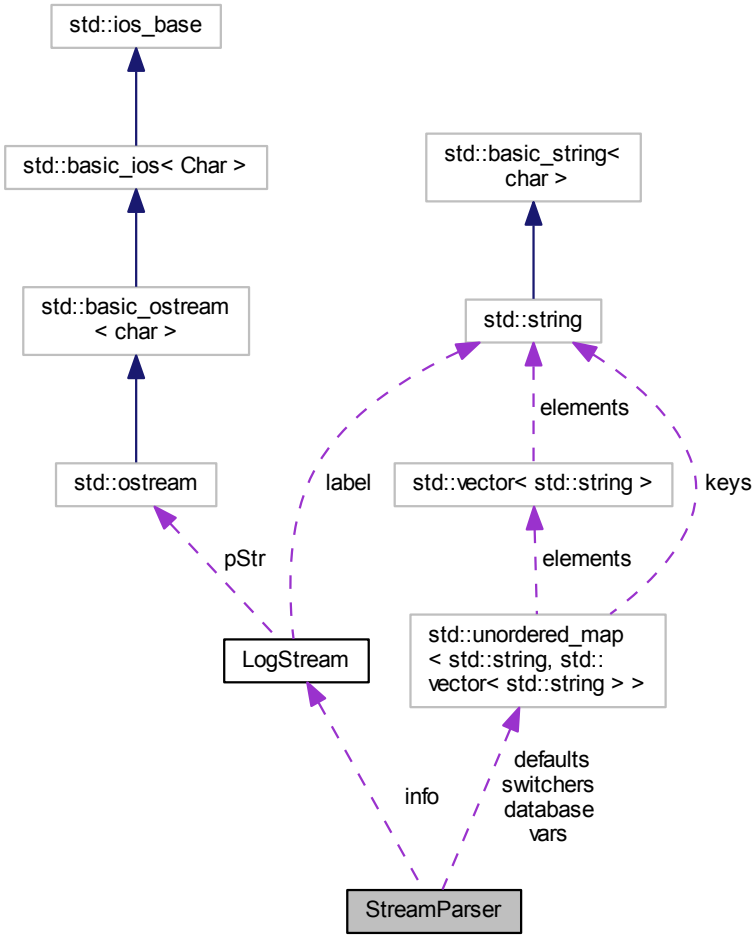
- Sheet/Sheet.h
- Sheet/Sheet.cpp

10.26 Класс StreamParser

Класс, позволяющий выполнять разбор файлов и строк с настройками и параметрами

#include <StreamParser.h>

Граф связей класса StreamParser:



Открытые члены

- StreamParser (LogStream &infoStream, const std::string &label, std::istream &mainStream, std↩ ::istream &defaultsStream, std::istream &switchersStream, std::istream &varsStream, std::vector< std::string > specificKey={})

  Конструктор, принимающий четыре потока
- StreamParser (LogStream &infoStream, const std::string &label, std::istream &mainStream, std↩ ::istream &defaultsStream, std::vector< std::string > specificKey={})

  Конструктор, принимающий два потока
- StreamParser (LogStream &infoStream, const std::string &label, std::istream &mainStream, char openBracket='{', char closeBracket='}')

  Конструктор, принимающий один поток
- ~StreamParser ()

  Деструктор
- template<typename T >
  void SetDefault (const std::string &name, T &res, const T *defValue) const

  Установка значения параметра по умолчанию

- void get (const std::string &name, std::vector< Point2D > &res, const std::vector< Point2D > *defValue=nullptr) const

  Считывание вектора из двумерных точек из базы данных
- void get (const std::string &name, std::vector< Vortex2D > &res, const std::vector< Vortex2D > *defValue=nullptr) const

  Считывание вектора из вихрей из базы данных
- template<typename T >
  void get (const std::string &name, std::vector< T > &res, const std::vector< T > *def↩Value=nullptr) const

  Считывание вектора из простых типов из базы данных
- template<typename T >
  void get (const std::string &name, T &res, const T *defValue=nullptr) const

  Считывание скаляра из базы данных
- void get (const std::string &name, Point2D &res, const Point2D *defValue=nullptr) const

  Считывание точки из базы данных
- void get (const std::string &name, bool &res, const bool *defValue=nullptr) const

  Считывание логической переменной из базы данных

Открытые статические члены

- static std::string UpperCase (const std::string &line)

  Перевод строки в верхний регистр
- static std::vector< std::string > StringToVector (std::string line, char openBracket='(', char close↩Bracket=')')

  Разбор строки, содержащей запятые, на отдельные строки
- static std::string VectorStringToString (const std::vector< std::string > &_vecString)

  Объединение вектора (списка) из строк в одну строку
- static std::pair< std::string, std::string > SplitString (LogStream &info, std::string line, bool upcase=true)

  Разбор строки на пару ключ-значение

Закрытые члены

- void ParseStream (std::istream &stream, std::unordered_map< std::string, std::vector< std::string >> &database, std::vector< std::string > specificKey={}, bool replaceVars=false, char open↩Bracket='(', char closeBracket=')')

  Парсинг заданного потока
- void ReplaceVarsInString (std::string &st)

  Замена переменных в строке их значениями

Закрытые данные

- LogStream info

  Поток для вывода логов и сообщений об ошибках
- std::unordered_map< std::string, std::vector< std::string > > database

  Данные считываемые из основного файла-паспорта расчета
- std::unordered_map< std::string, std::vector< std::string > > defaults

  Данные считываемые из списка умолчаний
- std::unordered_map< std::string, std::vector< std::string > > vars

  Данные считываемые из параметров конкретной задачи
- std::unordered_map< std::string, std::vector< std::string > > switchers

  Данные считываемые из перечня параметров-переключателей

10.26.1 Подробное описание

Класс, позволяющий выполнять разбор файлов и строк с настройками и параметрами

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле StreamParser.h строка 64

10.26.2 Конструктор(ы)

10.26.2.1 StreamParser() [1/3]

```
StreamParser::StreamParser (
            LogStream & infoStream,
            const std::string & label,
            std::istream & mainStream,
            std::istream & defaultsStream,
            std::istream & switchersStream,
            std::istream & varsStream,
            std::vector< std::string > specificKey = {} )
```

Конструктор, принимающий четыре потока

Заполняет 4 базы данных — database, defaults, switchers, vars
Используется для установки основных параметров расчета

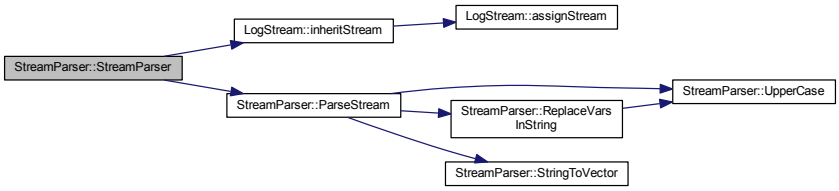
Аргументы

in,out	infoStream	базовый поток для вывода логов
in	label	константная ссылка на метку парсера для вывода логов
in	mainStream	ссылка на основной поток параметров
in	defaultsStream	ссылка на поток по умолчанию
in	switchersStream	ссылка на поток параметров-переключателей
in	varsStream	ссылка на поток переменных
in	specificKey	поиск только ключей из данного списка (по умолчанию — пустой список, что означает разбор всех ключей)

См. определение в файле StreamParser.cpp строка 42

```
43 {
44     info.inheritStream(infoStream, label);
45
46     ParseStream(varsStream, vars);
47     ParseStream(defaultsStream, defaults);
48     ParseStream(switchersStream, switchers);
49     ParseStream(mainStream, database, specificKey, true);
50 }//StreamParser(...)
```

Граф вызовов:



10.26.2.2 StreamParser() [2/3]

```
StreamParser::StreamParser (
            LogStream & infoStream,
            const std::string & label,
            std::istream & mainStream,
            std::istream & defaultsStream,
            std::vector< std::string > specificKey = {} )
```

Конструктор, принимающий два потока

Заполняет 2 базы данных — database, defaults; базы данных switchers и vars остаются пустыми Используется для первоначального считывания параметров задач из списка задач и формирования очереди

Аргументы

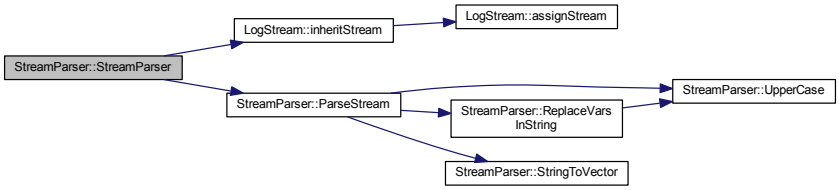
in,out	infoStream	базовый поток для вывода логов
in	label	константная ссылка на метку парсера для вывода логов
in	mainStream	ссылка на основной поток параметров
in	defaultsStream	ссылка на поток по умолчанию
in	specificKey	поиск только ключей из данного списка (по умолчанию — пустой список, что означает разбор всех ключей)

См. определение в файле StreamParser.cpp строка 54

```
55 {
56     info.inheritStream(infoStream, label);
```

```
57
58     ParseStream(mainStream, database, specificKey);
59     ParseStream(defaultsStream, defaults);
60
61     switchers.clear();
62     vars.clear();
63 }//StreamParser(...)
```

Граф вызовов:



10.26.2.3 StreamParser() [3/3]

```
StreamParser::StreamParser (
            LogStream & infoStream,
            const std::string & label,
            std::istream & mainStream,
            char openBracket = '{',
            char closeBracket = '}' )
```

Конструктор, принимающий один поток

Заполняет 1 базу данных — database; базы данных defaults, switchers и vars остаются пустыми
Используется для считывания профиля, следа, списка задач

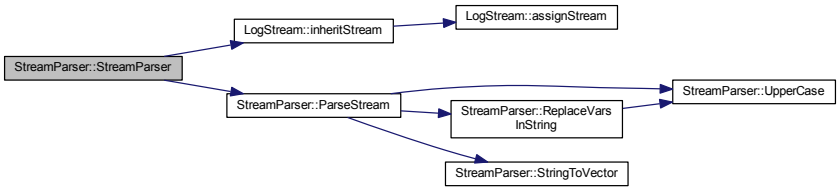
Аргументы

in,out	infoStream	базовый поток для вывода логов
in	label	константная ссылка на метку парсера для вывода логов
in	mainStream	ссылка на основной поток параметров
in	openBracket	тип открывающейся скобки (по умолчанию — "{")
in	closeBracket	тип закрывающейся скобки (по умолчанию — "}")

См. определение в файле StreamParser.cpp строка 67

```
68 {
69     info.inheritStream(infoStream, label);
70
71     ParseStream(mainStream, database, {}, false, openBracket, closeBracket);
72
73     defaults.clear();
74     switchers.clear();
75     vars.clear();
76 }//StreamParser(...)
```

Граф вызовов:



10.26.2.4 ~StreamParser()

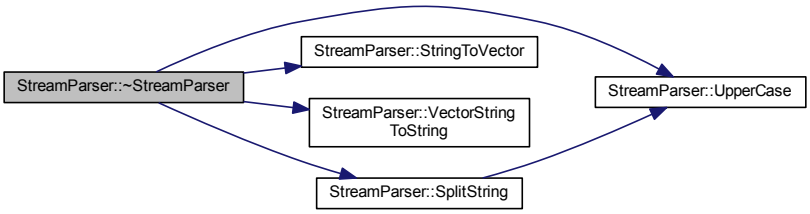
StreamParser::~StreamParser ( ) [inline]

Деструктор

См. определение в файле StreamParser.h строка 155

```
155 { };
```

Граф вызовов:



10.26.3 Методы

10.26.3.1 get() [1/6]

```
void StreamParser::get (
            const std::string & name,
            std::vector< Point2D > & res,
            const std::vector< Point2D > * defValue = nullptr ) const [inline]
```

Считывание вектора из двумерных точек из базы данных

Переопределение метода get() для ситывания вектора из точек (Point2D) из базы данных

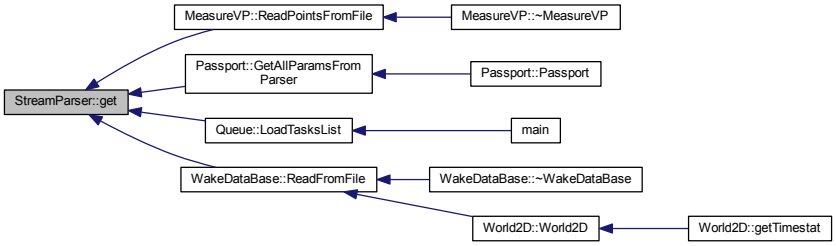
Аргументы

in	name	константная ссылка на строку — имя считываемого параметра
out	res	ссылка на значение, считываемое из базы данных
in	defValue	указатель на константу — значение по умолчанию (по умолчанию nullptr)

См. определение в файле StreamParser.h строка 216

```
217     {
218         res.resize(0);
219         std::unordered_map<std::string, std::vector<std::string>>::const_iterator search_it;
220
221         //поиск ключа в базе и, если нет, то в базе умолчаний
222         if (((search_it = database.find(UpperCase(name))) !=
    database.end()) ||
223             ((search_it = defaults.find(UpperCase(name))) !=
    defaults.end()))
224         {
225             for (size_t i = 0; i < (search_it->second).size(); ++i)
226             {
227                 std::string s = (search_it->second)[i];
228
229                 Point2D elem;
230                 size_t pos = s.find(',', 1);
231                 std::stringstream(s.substr(1, pos)) >> elem[0];
232                 std::stringstream(s.substr(pos + 1, s.length() - pos)) >> elem[1];
233                 res.push_back(elem);
234             }
235         }
236         //else
237         //  SetDefault(name, res, defValue);
238     };
```

Граф вызова функции:



10.26.3.2 get() [2/6]

void StreamParser::get (
const std::string & name,
std::vector< Vortex2D > & res,
const std::vector< Vortex2D > * defValue = nullptr) const [inline]

Считывание вектора из вихрей из базы данных

Переопределение метода get() для ситывания вектора из вихрей (Vortex2D) из базы данных

Аргументы

in	name	константная ссылка на строку — имя считываемого параметра
out	res	ссылка на вектор из вихрей, считываемый из базы данных
in	defValue	указатель на константу — значение по умолчанию (по умолчанию nullptr)

См. определение в файле StreamParser.h строка 247

```
248    {
249        res.resize(0);
250        std::unordered_map<std::string, std::vector<std::string>>::const_iterator search_it;
251
252        //поиск ключа в базе и, если нет, то в базе умолчаний
253        if (((search_it = database.find(UpperCase(name))) !=
   database.end()) ||
254            ((search_it = defaults.find(UpperCase(name))) !=
   defaults.end()))
255        {
256            for (size_t i = 0; i < (search_it->second).size(); ++i)
257            {
258                std::string s = (search_it->second)[i];
259
260                Point2D r;
261                double g;
262                size_t pos1 = s.find(',', 1);
263                size_t pos2 = s.find(',', pos1 + 1);
264                std::stringstream(s.substr(1, pos1)) >> r[0];
265                std::stringstream(s.substr(pos1 + 1, s.length() - pos1)) >> r[1];
266                std::stringstream(s.substr(pos2 + 1, s.length() - pos2)) >> g;
267
268                Vortex2D elem(r, g);
269
270                res.push_back(elem);
271            }
272        }
273        //else
274        //  SetDefault(name, res, defValue);
275    };
```

10.26.3.3 get() [3/6]

```
template<typename T >
void StreamParser::get (
            const std::string & name,
            std::vector< T > & res,
            const std::vector< T > * defValue = nullptr ) const   [inline]
```

Считывание вектора из простых типов из базы данных

Шаблонный метод get() для считывания вектора из простых типов из базы данных

Template Parameters

T	тип считываемых данных

Аргументы

in	name	константная ссылка на строку — имя считываемого параметра
out	res	ссылка на вектор из данных, считываемый из базы данных

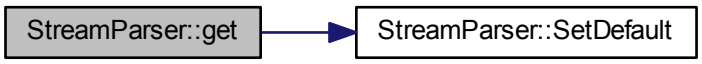
Аргументы

in	defValue	указатель на константу — значение по умолчанию (по умолчанию nullptr)

См. определение в файле StreamParser.h строка 285

```
286     {
287         res.resize(0);
288         std::unordered_map<std::string, std::vector<std::string>>::const_iterator search_it;
289
290         //поиск ключа в базе и, если нет, то в базе умолчаний
291         if ( ((search_it = database.find(UpperCase(name))) !=
    database.end()) ||
292             ((search_it = defaults.find(UpperCase(name))) !=
    defaults.end()))
293         {
294             for (size_t i = 0; i < (search_it->second).size(); ++i)
295             {
296                 std::string s = (search_it->second)[i];
297
298                 std::stringstream ss(s);
299                 T elem;
300                 ss >> elem;
301                 if (typeid(elem).name() == typeid(std::string("TestString")).name())
302                 {
303                     std::string* str = reinterpret_cast<std::string*>(&elem);
304                     str->erase(remove(str->begin(), str->end(), '\"'), str->end());
305                 }
306                 res.push_back(elem);
307             }
308         }
309         else
310             SetDefault(name, res, defValue);
311     };
```

Граф вызовов:



10.26.3.4 get() [4/6]

```
template<typename T >
void StreamParser::get (
            const std::string & name,
            T & res,
            const T * defValue = nullptr ) const    [inline]
```

Считывание скаляра из базы данных

Шаблонный метод get() для считывания скаляра из базы данных

Template Parameters

T	тип считываемых данных

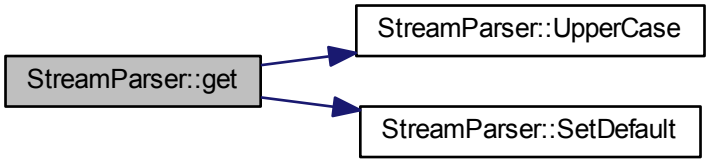
Аргументы

in	name	константная ссылка на строку — имя считываемого параметра
out	res	ссылка данные, считываемые из базы данных
in	defValue	указатель на константу — значение по умолчанию (по умолчанию nullptr)

См. определение в файле StreamParser.h строка 321

```
322     {
323         std::unordered_map<std::string, std::vector<std::string>>::const_iterator search_it;
324
325         //поиск ключа в базе и, если нет, то в базе умолчаний
326         if ( (( (search_it = database.find(UpperCase(name))) != database.end()) && ((search_it->second).size() > 0)) ||
327             (( (search_it = defaults.find(UpperCase(name))) != defaults.end()) && ((search_it->second).size() > 0)) )
328         {
329             if ((search_it->second).size() == 1)
330             {
331                 std::string s = (search_it->second)[0];
332
333                 //проверка на значение-переключатель
334                 if (((search_it = switchers.find(UpperCase(s))) != switchers.end()) && ((search_it->second).size() > 0))
335                     s = search_it->second[0];
336
337                 std::stringstream ss(s);
338                 T elem;
339                 ss >> elem;
340                 if (typeid(elem).name() == typeid(std::string("TestString")).name())
341                 {
342                     std::string* str = reinterpret_cast<std::string*>(&elem);
343                     str->erase(remove(str->begin(), str->end(), '\"'), str->end());
344                 }
345                 res = elem;
346             }
347             else
348             {
349                 info('e') << "parameter " << name << " is list (only scalar is available)" << std::endl;
350                 exit(-1);
351             }
352         }
353         else
354             SetDefault(name, res, defValue);
355     };
```

Граф вызовов:



10.26.3.5 get() [5/6]

```
void StreamParser::get (
            const std::string & name,
            Point2D & res,
            const Point2D * defValue = nullptr ) const   [inline]
```

Считывание точки из базы данных

Переопределение метода get() для ситывания точки (Point2D) из базы данных

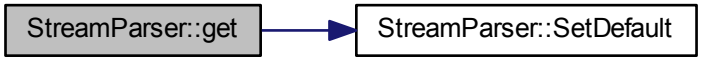
Аргументы

in	name	константная ссылка на строку — имя считываемого параметра
out	res	ссылка на точку, считываемую из базы данных
in	defValue	указатель на константу — значение по умолчанию (по умолчанию nullptr)

См. определение в файле StreamParser.h строка 363

```
364     {
365         std::unordered_map<std::string, std::vector<std::string>>::const_iterator search_it;
366
367         //поиск ключа в базе и, если нет, то в базе умолчаний
368         if ( ((search_it = database.find(UpperCase(name))) !=
    database.end()) ||
369              ((search_it = defaults.find(UpperCase(name))) !=
    defaults.end()) )
370         {
371             if ((search_it->second).size() == res.size())
372             {
373                 for (size_t i = 0; i < (search_it->second).size(); ++i)
374                 {
375                     std::string s = (search_it->second)[i];
376
377                     std::stringstream ss(s);
378                     double elem;
379                     ss >> elem;
380                     res[i] = elem;
381                 }
382             }
383             else
384             {
385                 info('e') << "parameter " << name << " length differs from 2 (only Point2D is
    available)" << std::endl;
386                 exit(-1);
387             }
388
389         }
390         else
391             SetDefault(name, res, defValue);
392     }
```

Граф вызовов:



10.26.3.6 get() [6/6]

```
void StreamParser::get (
            const std::string & name,
            bool & res,
            const bool * defValue = nullptr ) const   [inline]
```

Считывание логической переменной из базы данных

Переопределение метода get() для ситывания логической переменной (bool) из базы данных

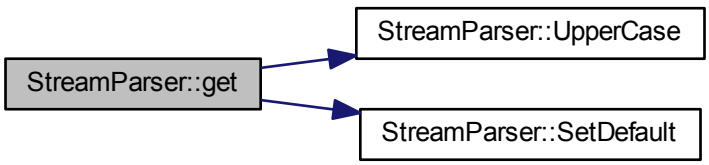
Аргументы

in	name	константная ссылка на строку — имя считываемого параметра
out	res	ссылка на логическую переменную, считываемую из базы данных
in	defValue	указатель на константу — значение по умолчанию (по умолчанию nullptr)

См. определение в файле StreamParser.h строка 401

```
402   {
403       std::unordered_map<std::string, std::vector<std::string>>::const_iterator search_it;
404
405       //поиск ключа в базе и, если нет, то в базе умолчаний
406       if ((((search_it = database.find(UpperCase(name))) !=
database.end()) && ((search_it->second).size() > 0)) ||
407           (((search_it = defaults.find(UpperCase(name))) !=
defaults.end()) && ((search_it->second).size() > 0)))
408       {
409           if ((search_it->second).size() == 1)
410           {
411               std::string s = (search_it->second)[0];
412
413               //проверка на значение-переключатель
414               if (((search_it = switchers.find(UpperCase(s))) !=
switchers.end()) && ((search_it->second).size() > 0))
415                   s = search_it->second[0];
416
417               if ((UpperCase(s) == "FALSE") || (UpperCase(s) == "NO") || (s.c_str()[0]
== '0'))
418                   res = false;
419               else
420                   res = true;
421           }
422           else
423           {
424               info('e') << "parameter " << name << " is list (only scalar is available)" << std::endl
;
425               exit(-1);
426           }
427       }
428       else
429           SetDefault(name, res, defValue);
430   };
```

Граф вызовов:



10.26.3.7 ParseStream()

```
void StreamParser::ParseStream (
            std::istream & stream,
            std::unordered_map< std::string, std::vector< std::string >> & database,
            std::vector< std::string > specificKey = {},
            bool replaceVars = false,
            char openBracket = '(',
            char closeBracket = ')' )   [private]
```

Парсинг заданного потока

Разбирает поток вида key = value или key = { value1, value2, ..., valueN }
Возвращает базу данных, указывается тип скобок, по умолчанию — круглые

Аргументы

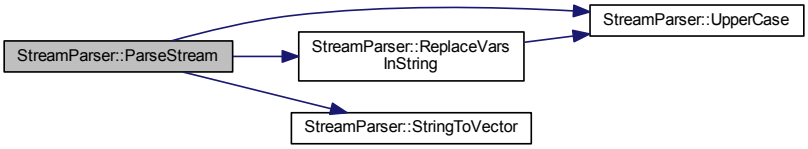
in	stream	ссылка на поток, который требуется распарсить
out	database	ссылка на заполняемую базу данных (unordered_map)
in	specificKey	поиск только ключей из данного списка (по умолчанию — пустой список, что означает разбор всех ключей)
in	replaceVars	признак замены переменных из базы vars (по умолчанию false)
in	openBracket	тип открывающейся скобки (по умолчанию — "(")
in	closeBracket	тип закрывающейся скобки (по умолчанию — ")")

См. определение в файле StreamParser.cpp строка 174

```
175 {
176     std::string line, readline;
177
178     while (stream.good())
179     {
180         line = "";
181
182         do
183         {
184             getline(stream, readline);
185
186             size_t posComment = std::min(readline.find('#'), readline.find("//"));
187             //min, так как если не найдено -- то find возвращает string::npos, которое равно (-1), но
    приводится к большому положительному size_t
188             if (posComment != std::string::npos)
189                 readline = readline.substr(0, posComment);
190
191             line += readline;
192
193             readline.erase(remove(readline.begin(), readline.end(), ' '), readline.end());
194             readline.erase(remove(readline.begin(), readline.end(), '\t'), readline.end());
195
196         } while ( (readline.size()>0) && (readline[readline.size() - 1] == '\\') && (stream.good()));
197
198         line.erase(remove(line.begin(), line.end(), '\\'), line.end());
199         line.erase(remove(line.begin(), line.end(), ' '), line.end());
200         line.erase(remove(line.begin(), line.end(), '\t'), line.end());
201
202         size_t posEqual = line.find('=');
203         if (posEqual != -1)
204         {
205             std::string key = line.substr(0, posEqual);
206             auto search_it = database.find(UpperCase(key));
207
208             if (search_it == database.end())
```

```
209         {
210             if ((specificKey.size() == 0) || (std::find(specificKey.begin(),specificKey.end(),key) !=
    specificKey.end()) )
211             {
212                 std::string value = line.substr(posEqual + 1, line.length());
213                 if (replaceVars)
214                     ReplaceVarsInString(value);
215                 database.insert(make_pair(UpperCase(key),
    StringToVector(value, openBracket, closeBracket)));
216             }
217         }
218         else
219         {
220             info('e') << "key <" << key << "> is found twice" << std::endl;
221             exit(-1);
222         }
223     }//if (posEqual != -1)
224   }//while (streamName.good())
225
226   //Указатель возвращаем в начало потока
227   stream.clear();
228   stream.seekg(0, std::ios::beg);
229 }//ParseStream(...)
```

Граф вызовов:



Граф вызова функции:

10.26.3.8 ReplaceVarsInString()

void StreamParser::ReplaceVarsInString (
std::string & st) [private]

Замена переменных в строке их значениями

Находит в строке все переменные, имена которых начинаются с символа $, и заменяет их на значения, извлекаемые из базы данных vars

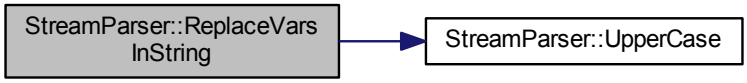
Аргументы

in	st	строка, которая подлежит обработке

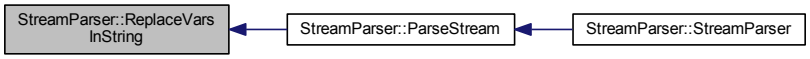
См. определение в файле StreamParser.cpp строка 233

```
234 {
235     while (st.find("$") != -1)
236     {
237         size_t startVar = st.find('$');
238
239         size_t endVar = -1 + std::min({
240             st.find(" ", startVar + 1),
241             st.find(",", startVar + 1),
242             st.find(";", startVar + 1),
243             st.find("$", startVar + 1),
244             st.find("(", startVar + 1),
245             st.find(")", startVar + 1),
246             st.find("{", startVar + 1),
247             st.find("}", startVar + 1),
248             st.find("\n", startVar + 1),
249             st.length()
250         });
251
252         std::string var = st.substr(startVar + 1, endVar - startVar);
253
254         auto search_var = vars.find(UpperCase(var));
255         if ((search_var != vars.end()) && ((search_var->second).size() > 0))
256         {
257             std::vector<std::string> findString = search_var->second;
258
259             std::stringstream ss;
260
261             ss << st.substr(0, startVar);
262
263             if (findString.size() == 1)
264             {
265                 ss << findString[0];
266             }
267             else
268             {
269                 ss << "{" << findString[0];
270                 if (findString.size() > 0)
271                 {
272                     for (size_t sz = 1; sz < findString.size(); ++sz)
273                         ss << "," << findString[sz];
274                 }
275                 ss << "}";
276             }
277
278
279             size_t startP = endVar+1;
280             size_t endP = st.length() - 1;
281
282             if (startP <= endP)
283                 ss << st.substr(startP, endP);
284
285             st = ss.str();
286         }
287         else
288         {
289             info('e') << "substitution $" << var << " is us undefined" << std::endl;
290             exit(1);
291         }
292     }
293 }
```

Граф вызовов:



Граф вызова функции:



10.26.3.9 SetDefault()

```
template<typename T >
void StreamParser::SetDefault (
            const std::string & name,
            T & res,
            const T * defValue ) const   [inline]
```

Установка значения параметра по умолчанию

Устанавливает параметру значение по умолчанию, если оно задано в виде константной ссылки

Template Parameters

T	тип параметра, значение которого устанавливается

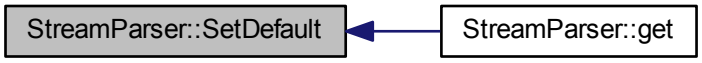
Аргументы

in	name	константная ссылка на строку — имя устанавливаемого параметра (необходимо только для отображения в лог)
in	res	ссылка на задаваемый параметр
in	defValue	указатель на константу — значение по умолчанию

См. определение в файле StreamParser.h строка 196

```
197     {
198         if (defValue != nullptr)
199         {
200             res = *defValue;
201             info('i') << "parameter <" << name << " = " << res << "> set as default" << std::endl;
202         }
203         else
204         {
205             info('e') << "parameter " << name << " is not found" << std::endl;
206             exit(-1);
207         }
208     }
```

Граф вызова функции:



10.26.3.10 SplitString()

```
std::pair< std::string, std::string > StreamParser::SplitString (
            LogStream & info,
            std::string line,
            bool upcase = true )   [static]
```

Разбор строки на пару ключ-значение

Разбирает строку вида xxx(yyy) на пару подстрок xxx и yyy

Аргументы

in,out	info	поток для вывода логов и сообщений об ошибках
in	line	разбираемая строка вида xxx(yyy)
in	upcase	признак перевода ключа в верхний регистр

Возвращает

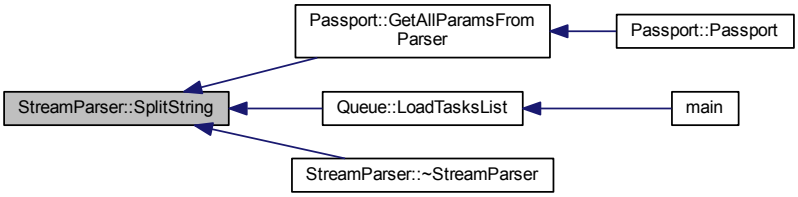
пара строк (xxx, yyy)

См. определение в файле StreamParser.cpp строка 156

```
157 {
158     size_t posBegin = line.find('(', 0);
159     size_t posEnd = line.find(')', 0);
160     if ((posBegin != -1) && (posEnd == -1))
161     {
162         info('e') << "error while parsing line " << line << std::endl;
163         exit(-1);
164     }
165     else
166     {
167         std::string sub = line.substr(0, posBegin);
168         return std::pair<std::string, std::string>(upcase ? UpperCase(sub) : sub, "{" + line.
    substr(posBegin + 1, posEnd - posBegin - 1) + "}");
169     }
170 }//SplitString(...)
```

Граф вызовов:

Граф вызова функции:



10.26.3.11 StringToVector()

```
std::vector< std::string > StreamParser::StringToVector (
            std::string line,
            char openBracket = '(',
            char closeBracket = ')' )   [static]
```

Разбор строки, содержащей запятые, на отдельные строки

Запятые внутри парных скобок игнорируются и не воспринимаются как разделители

Аргументы

in	line	разбираемая строка
in	openBracket	тип открывающейся скобки (по умолчанию — "(")
in	closeBracket	тип закрывающейся скобки (по умолчанию — ")")

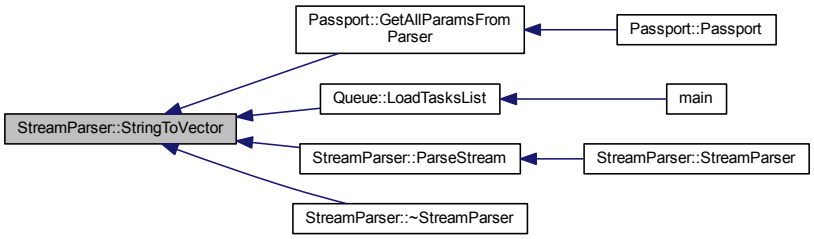
Возвращает

список (вектор) из отдельных строк

См. определение в файле StreamParser.cpp строка 98

```
{
    //LogStream info;
    //info.inheritStream(infoStream, label);

    if (line[0] != '{')
    {
        if (line.length() > 0)
            return { line };
        else
        {
            //info('i') << "empty parameter" << std::endl;
            return std::vector<std::string>({});
        }
    }
    else
    {
        if (line[1] != '}')
        {
            std::vector<std::string> vecLine;
            size_t pos;
            while ( ((pos = line.find(',', 0)) < line.find(openBracket, 1)) ||
                ( (pos = line.find(',', std::max(1, static_cast<int>(line.find(closeBracket, 1)) ))) !=
std::string::npos) ||
                    (pos = line.length()-1)  )
            {
                std::string subline = line.substr(1, pos - 1);
                if (subline != "")
                    vecLine.push_back(subline);

                line.erase(1, subline.size() + 1);
            }
            return vecLine;
        }
        else
        {
            //info('i') << "empty parameter list" << std::endl;
            return std::vector<std::string>({});
        }//else if (line[1] != '}')
    }//else if (st[0] != '{')
}//StringToVector(...)
```

Граф вызова функции:



10.26.3.12 UpperCase()

```
std::string StreamParser::UpperCase (
            const std::string & line )   [static]
```

Перевод строки в верхний регистр

Аргументы

in	line	константная ссылка на обрабатываемую строку

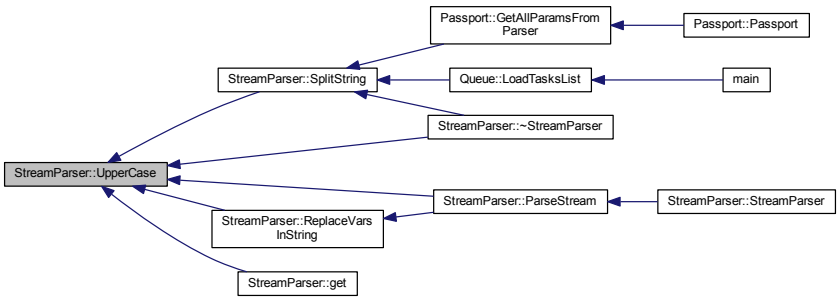
Возвращает

строка в верхнем регистре

См. определение в файле StreamParser.cpp строка 79

```
80 {
81     std::string str(line);
82     bool flag = true;
83
84     for (auto & c : str)
85     {
86         if (c == '"')
87             flag = !flag;
88         if (flag)
89             c = toupper(c);
90     }
91
92     //std::transform(str.begin(), str.end(), str.begin(), ::toupper);
93     return str;
94 };//UpperCase(...)
```

Граф вызова функции:



10.26.3.13 VectorStringToString()

std::string StreamParser::VectorStringToString (
const std::vector< std::string > & _vecString) [static]

Объединение вектора (списка) из строк в одну строку

После каждой строки вставляется символ конца строки

Аргументы

in	_vecString	константная ссылка на вектор из строк

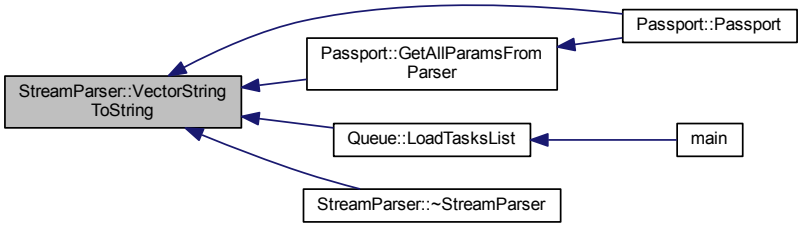
Возвращает

сплошную строку с символами конца строки между подстроками

См. определение в файле StreamParser.cpp строка 142

```
{
    std::string buf;
    for (size_t q = 0; q < _vecString.size(); ++q)
    {
        buf.append(_vecString[q]);
        buf.append("\n");
    }

    return buf;
}//VectorStringToString(...)
```

Граф вызова функции:



10.26.4 Данные класса

10.26.4.1 database

std::unordered_map<std::string, std::vector<std::string> > StreamParser::database [private]

Данные считываемые из основного файла-паспорта расчета

Неупорядоченный ассоциативный контейнер, хранящий данные из паспорта расчета
По умолчанию данные берутся из файла passport, но это имя может быть изменено ключом pspFile в строке параметров задачи

См. определение в файле StreamParser.h строка 75

10.26.4.2 defaults

std::unordered_map<std::string, std::vector<std::string> > StreamParser::defaults [private]

Данные считываемые из списка умолчаний

Неупорядоченный ассоциативный контейнер, хранящий данные из файла со списком значений по умолчанию
По умолчанию данные берутся из файла defaults

См. определение в файле StreamParser.h строка 81

10.26.4.3 info

LogStream StreamParser::info [mutable], [private]

Поток для вывода логов и сообщений об ошибках

См. определение в файле StreamParser.h строка 69

10.26.4.4 switchers

std::unordered_map<std::string, std::vector<std::string> > StreamParser::switchers [private]

Данные считываемые из перечня параметров-переключателей

Неупорядоченный ассоциативный контейнер, хранящий значения параметров-переключателей
По умолчанию данные берутся из файла switcher.

См. определение в файле StreamParser.h строка 93

10.26.4.5 vars

std::unordered_map<std::string, std::vector<std::string> > StreamParser::vars [private]

Данные считываемые из параметров конкретной задачи

Неупорядоченный ассоциативный контейнер, хранящий данные из строки — перечня параметров конкретной задачи в файле problems.
Здесь же определяются переменные, которые затем могут использоваться методом разыменовывания в файлах-паспортах конкретных задач

См. определение в файле StreamParser.h строка 87

Объявления и описания членов классов находятся в файлах:

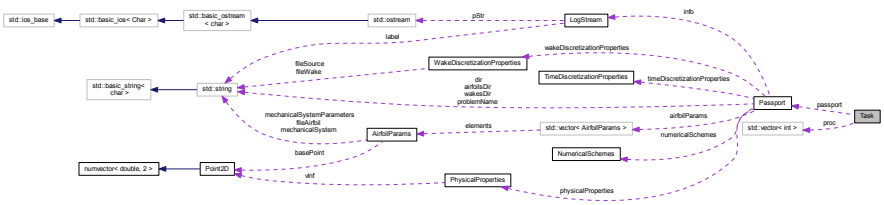
- StreamParser/StreamParser.h
- StreamParser/StreamParser.cpp

10.27 Класс Task

Класс, опеделяющий описание каждой задачи в смысле прохождения процесса ее решения

#include <Task.h>

Граф связей класса Task:



Открытые члены

- Task (const Passport &_passport)

  Конструктор
- ~Task ()

  Деструктор

Открытые атрибуты

- int nProc

  Число процессоров
- std::vector< int > proc

  Номера процессоров, занятых решением данной задачи
- TaskState state

  Флаг состояния задачи
- std::pair< int, int > startEndKvant

  Номера квантов времени, в которых начался и завершился счет задачи
- double wTime

  Время в секундах, потраченное на решение данной задачи
- Passport passport

  Паспорт задачи

10.27.1 Подробное описание

Класс, опеделяющий описание каждой задачи в смысле прохождения процесса ее решения

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Task.h строка 89

10.27.2 Конструктор(ы)

10.27.2.1 Task()

```
Task::Task (
            const Passport & _passport ) [inline]
```

Конструктор

Аргументы

_passport	константная ссылка на паспорт задачи

См. определение в файле Task.h строка 96

```
97        : passport(_passport) { };
```

10.27.2.2 ~Task()

Task::~Task ( ) [inline]

Деструктор

См. определение в файле Task.h строка 100

```
100 {};
```

10.27.3 Данные класса

10.27.3.1 nProc

int Task::nProc

Число процессоров

См. определение в файле Task.h строка 100

10.27.3.2 passport

Passport Task::passport

Паспорт задачи

См. определение в файле Task.h строка 122

10.27.3.3 proc

std::vector<int> Task::proc

Номера процессоров, занятых решением данной задачи

См. определение в файле Task.h строка 108

10.27.3.4 startEndKvant

std::pair<int, int> Task::startEndKvant

Номера квантов времени, в которых начался и завершился счет задачи

См. определение в файле Task.h строка 114

10.27.3.5 state

TaskState Task::state

Флаг состояния задачи

См. определение в файле Task.h строка 111

10.27.3.6 wTime

double Task::wTime

Время в секундах, потраченное на решение данной задачи

Заполняется после окончания решения задачи

См. определение в файле Task.h строка 119

Объявления и описания членов класса находятся в файле:

- Task/Task.h

10.28 Структура TimeDiscretizationProperties

Структура, задающая параметры процесса интегрирования по времени

#include <Passport.h>

Открытые атрибуты

- double timeStart

  Начальное время
- double timeStop

  Конечное время
- double dt

  Шаг по времени
- int nameLength

  Число разрядов в имени файла
- int saveTXT

  Шаг сохранения кадров в текстовые файлы
- int saveVTK

  Шаг сохранения кадров в бинарные файлы
- int saveVP

  Шаг вычисления и сохранения скорости и давления

10.28.1 Подробное описание

Структура, задающая параметры процесса интегрирования по времени

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Passport.h строка 137

10.28.2 Данные класса

10.28.2.1 dt

double TimeDiscretizationProperties::dt

Шаг по времени

См. определение в файле Passport.h строка 146

10.28.2.2 nameLength

int TimeDiscretizationProperties::nameLength

Число разрядов в имени файла

См. определение в файле Passport.h строка 149

10.28.2.3 saveTXT

int TimeDiscretizationProperties::saveTXT

Шаг сохранения кадров в текстовые файлы

См. определение в файле Passport.h строка 152

10.28.2.4 saveVP

int TimeDiscretizationProperties::saveVP

Шаг вычисления и сохранения скорости и давления

См. определение в файле Passport.h строка 158

10.28.2.5 saveVTK

int TimeDiscretizationProperties::saveVTK

Шаг сохранения кадров в бинарные файлы

См. определение в файле Passport.h строка 155

10.28.2.6 timeStart

double TimeDiscretizationProperties::timeStart

Начальное время

См. определение в файле Passport.h строка 140

10.28.2.7 timeStop

double TimeDiscretizationProperties::timeStop

Конечное время

См. определение в файле Passport.h строка 143

Объявления и описания членов структуры находятся в файле:

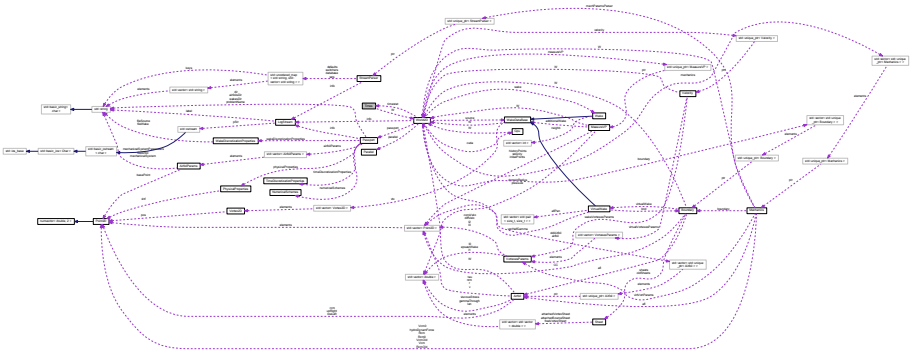
- Passport/Passport.h

10.29 Класс Times

Класс для сбора статистики времени исполнения основных шагов алгоритма и вывода ее в файл

#include <Times.h>

Граф связей класса Times:



Открытые члены

- Times (const World2D &W_)

  Конструктор
- ~Times ()

  Деструктор
- void GenerateStatHeader () const

  Генерация заголовка файла временной статистики
- void GenerateStatString () const

  Сохранение строки со статистикой в файл временной статистики
- void ToZero ()

  Обнуление состояния временной статистики

Открытые статические члены

- static double dT (const timePeriod &t)

Открытые атрибуты

- timePeriod timeWholeStep

  Начало и конец процесса выполнения шага целиком
- timePeriod timeReserveMemoryForMatrixAndRhs

  Начало и конец процесса выделения памяти под матрицу и правую часть
- timePeriod timeFillMatrixAndRhs

  Начало и конец процесса заполнения матрицы и формирования правой части
- timePeriod timeSolveLinearSystem

  Начало и конец процесса решения системы линейных алгебраических уравнений
- timePeriod timeCalcVortexConvVelo

  Начало и конец процесса вычисления конвективных скоростей вихрей
- timePeriod timeCalcVortexDiffVelo

  Начало и конец процесса вычисления диффузионных скоростей вихрей
- timePeriod timeGetHydroDynamForce

  Начало и конец процесса вычисления нагрузок
- timePeriod timeMoveVortexes

  Начало и конец процесса перемещения вихрей
- timePeriod timeCheckInside

  Начало и конец процесса контроля протыкания
- timePeriod timeRestruct

  Начало и конец процесса реструктуризации пелены
- timePeriod timeWakeSort

  Начало и конец процесса сортировки вихревого следа
- timePeriod timeSaveKadr

  Начало и конец процесса сохранения кадра в файл
- timePeriod timeVP

  Начало и конец процесса подсчета полей скоростей и давления и сохранения их в файл
- timePeriod timeOther

  Все прочее

Закрытые члены

- void ToZero (timePeriod &period)

Закрытые данные

- const World2D & W

  Константная ссылка на решаемую задачу

10.29.1 Подробное описание

Класс для сбора статистики времени исполнения основных шагов алгоритма и вывода ее в файл

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Times.h строка 54

10.29.2 Конструктор(ы)

10.29.2.1 Times()

Times::Times (
const World2D & W_) [inline]

Конструктор

См. определение в файле Times.h строка 112

```
113     : W(W_) {};
```

10.29.2.2 ~Times()

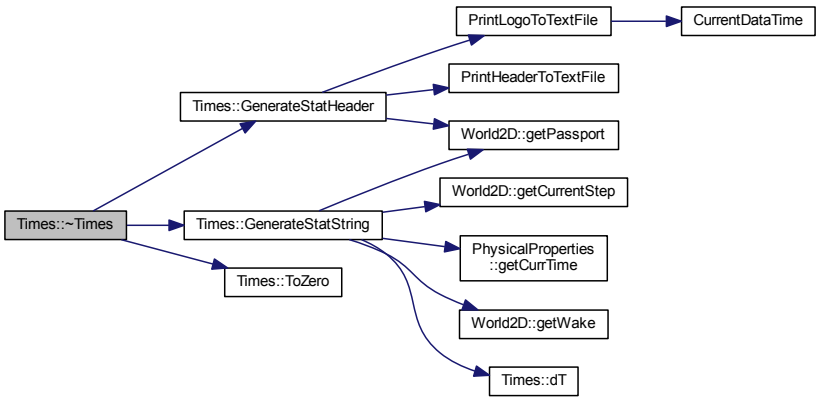
Times::~Times ( ) [inline]

Деструктор

См. определение в файле Times.h строка 116

```
116 {};
```

Граф вызовов:



10.29.3 Методы

10.29.3.1 dT()

static double Times::dT (
const timePeriod & t) [inline], [static]

Вычисление разницы во времени для пары засечек в секундах

Аргументы

in	t	константная ссылка на пару засечек времени

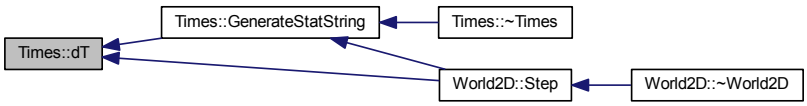
Возвращает

разницу в секундах

См. определение в файле Times.h строка 130

```
{
    return (t.second - t.first);
}//dT(...)
```

Граф вызова функции:



10.29.3.2 GenerateStatHeader()

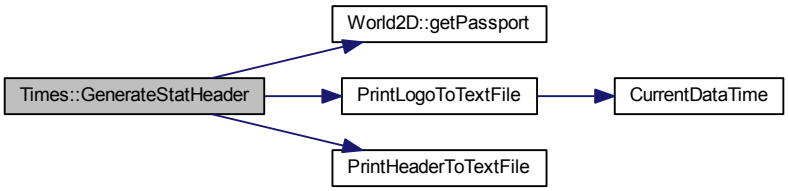
void Times::GenerateStatHeader ( ) const

Генерация заголовка файла временной статистики

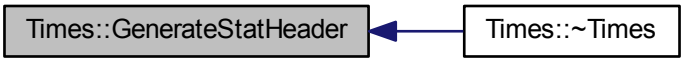
См. определение в файле Times.cpp строка 45

```
{
    std::stringstream timeStatFileName;
    timeStatFileName << W.getPassport().dir << "timestat";

    std::ofstream timeStatFile(timeStatFileName.str());
    PrintLogoToTextFile(timeStatFile, timeStatFileName.str(), "Time statistics");

    PrintHeaderToTextFile(timeStatFile, "step Time N tStep tMem tMatRhs tSolve tConvVelo tDiffVelo tForce tVelPres tMove tInside tRestr tWakeSort tSave tOther");

    timeStatFile.close();
    timeStatFile.clear();
}//GenerateStatHeader()
```

Граф вызовов:



Граф вызова функции:



10.29.3.3 GenerateStatString()

void Times::GenerateStatString ( ) const

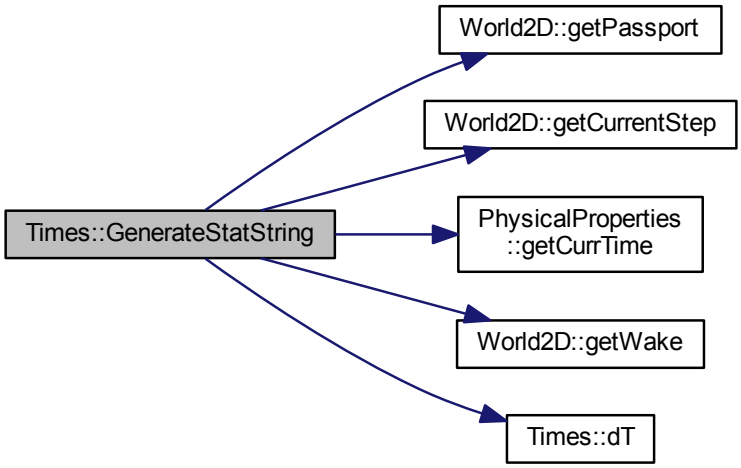
Сохранение строки со статистикой в файл временной статистики

См. определение в файле Times.cpp строка 62

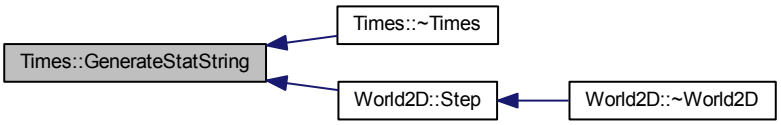
```
{
    std::ofstream timestatFile(W.getPassport().dir + "timestat", std::ios::app);

    timestatFile << std::endl
        << W.getCurrentStep() << "\t"
        << W.getPassport().physicalProperties.getCurrTime() << "\t"
        << W.getWake().vtx.size() << "\t"
        << dT(timeWholeStep) << "\t"
        << dT(timeReserveMemoryForMatrixAndRhs) << "\t"
```

```
        << dT(timeFillMatrixAndRhs) << "\t"
        << dT(timeSolveLinearSystem) << "\t"
        << dT(timeCalcVortexConvVelo) << "\t"
        << dT(timeCalcVortexDiffVelo) << "\t"
        << dT(timeGetHydroDynamForce) << "\t"
        << dT(timeVP) << "\t"
        << dT(timeMoveVortexes) << "\t"
        << dT(timeCheckInside) << "\t"
        << dT(timeRestruct) << "\t"
        << dT(timeWakeSort) << "\t"
        << dT(timeSaveKadr) << "\t"
        << dT(timeOther);

    timestatFile.close();
}//GenerateStatString()
```

Граф вызовов:



Граф вызова функции:



10.29.3.4 ToZero() [1/2]

void Times::ToZero (
timePeriod & period) [inline], [private]

Обнуление одного временного периода

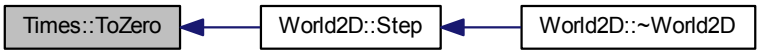
Аргументы

out	period	промежуток времени, начало и конец которого будут обнулены

См. определение в файле Times.h строка 62

```
63    {
64        period.first = 0;
65        period.second = 0;
66    }//ToZero(...)
```

Граф вызова функции:



10.29.3.5 ToZero() [2/2]

void Times::ToZero ( )

Обнуление состояния временной статистики

См. определение в файле Times.cpp строка 90

```
91 {
92     ToZero(timeWholeStep);
93     ToZero(timeReserveMemoryForMatrixAndRhs);
94     ToZero(timeFillMatrixAndRhs);
95     ToZero(timeSolveLinearSystem);
96     ToZero(timeCalcVortexConvVelo);
97     ToZero(timeCalcVortexDiffVelo);
98     ToZero(timeGetHydroDynamForce);
99     ToZero(timeVP);
100     ToZero(timeMoveVortexes);
101     ToZero(timeCheckInside);
102     ToZero(timeRestruct);
103     ToZero(timeWakeSort);
104     ToZero(timeSaveKadr);
105     ToZero(timeOther);
106 }
```

Граф вызова функции:

10.29.4 Данные класса

10.29.4.1 timeCalcVortexConvVelo

timePeriod Times::timeCalcVortexConvVelo

Начало и конец процесса вычисления конвективных скоростей вихрей

См. определение в файле Times.h строка 82

10.29.4.2 timeCalcVortexDiffVelo

timePeriod Times::timeCalcVortexDiffVelo

Начало и конец процесса вычисления диффузионных скоростей вихрей

См. определение в файле Times.h строка 85

10.29.4.3 timeCheckInside

timePeriod Times::timeCheckInside

Начало и конец процесса контроля протыкания

См. определение в файле Times.h строка 94

10.29.4.4 timeFillMatrixAndRhs

timePeriod Times::timeFillMatrixAndRhs

Начало и конец процесса заполнения матрицы и формирования правой части

См. определение в файле Times.h строка 76

10.29.4.5 timeGetHydroDynamForce

timePeriod Times::timeGetHydroDynamForce

Начало и конец процесса вычисления нагрузок

См. определение в файле Times.h строка 88

10.29.4.6 timeMoveVortexes

timePeriod Times::timeMoveVortexes

Начало и конец процесса перемещения вихрей

См. определение в файле Times.h строка 91

10.29.4.7 timeOther

timePeriod Times::timeOther

Все прочее

См. определение в файле Times.h строка 109

10.29.4.8 timeReserveMemoryForMatrixAndRhs

timePeriod Times::timeReserveMemoryForMatrixAndRhs

Начало и конец процесса выделения памяти под матрицу и правую часть

См. определение в файле Times.h строка 73

10.29.4.9 timeRestruct

timePeriod Times::timeRestruct

Начало и конец процесса реструктуризации пелены

См. определение в файле Times.h строка 97

10.29.4.10 timeSaveKadr

timePeriod Times::timeSaveKadr

Начало и конец процесса сохранения кадра в файл

См. определение в файле Times.h строка 103

10.29.4.11 timeSolveLinearSystem

timePeriod Times::timeSolveLinearSystem

Начало и конец процесса решения системы линейных алгебраических уравнений

См. определение в файле Times.h строка 79

10.29.4.12 timeVP

timePeriod Times::timeVP

Начало и конец процесса подсчета полей скоростей и давления и сохранения их в файл

См. определение в файле Times.h строка 106

10.29.4.13 timeWakeSort

timePeriod Times::timeWakeSort

Начало и конец процесса сортировки вихревого следа

См. определение в файле Times.h строка 100

10.29.4.14 timeWholeStep

timePeriod Times::timeWholeStep

Начало и конец процесса выполнения шага целиком

См. определение в файле Times.h строка 70

10.29.4.15 W

const World2D& Times::W [private]

Константная ссылка на решаемую задачу

См. определение в файле Times.h строка 58

Объявления и описания членов классов находятся в файлах:

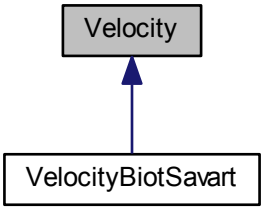
- Times/Times.h
- Times/Times.cpp

10.30 Класс Velocity

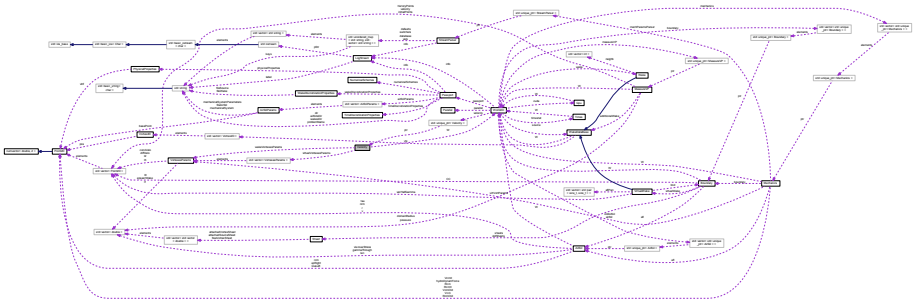
Абстрактный класс, определяющий способ вычисления скоростей

#include <Velocity.h>

Граф наследования:Velocity:



Граф связей класса Velocity:



Открытые члены

- Velocity (const World2D &W_)

  Конструктор
- virtual void CalcConvVeloToSetOfPoints (const WakeDataBase &pointsDb, std::vector< Point2D > &velo, std::vector< double > &domainRadius, bool onlyRadius=false)=0

  Вычисление конвективных скоростей и радиусов вихревых доменов в заданном наборе точек от следа
- void CalcConvVelo ()

  Вычисление конвективных скоростей вихрей и виртуальных вихрей в вихревом следе
- virtual void CalcDiffVeloI1I2ToSetOfPoints (const WakeDataBase &pointsDb, const std::vector< double > &domainRadius, const WakeDataBase &vorticesDb, std::vector< double > &I1, std↩ ::vector< Point2D > &I2)=0

  Вычисление числителей и знаменателей диффузионных скоростей в заданном наборе точек
- virtual void CalcDiffVeloI1I2ToSetOfPointsFromPanels (const WakeDataBase &pointsDb, const std::vector< double > &domainRadius, const Boundary &bnd, std::vector< double > &I1, std↩ ::vector< Point2D > &I2)=0
- void CalcDiffVelo ()

  Вычисление диффузионных скоростей вихрей и виртуальных вихрей в вихревом следе
- virtual ~Velocity ()

  Деструктор

Открытые атрибуты

- VortexesParams wakeVortexesParams

  Струтура, определяющая параметры вихрей в следе
- std::vector< VortexesParams > virtualVortexesParams

  Вектор струтур, определяющий параметры виртуальных вихрей для профилей

Защищенные данные

- const World2D & W

  Константная ссылка на решаемую задачу

10.30.1 Подробное описание

Абстрактный класс, определяющий способ вычисления скоростей

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Velocity.h строка 92

10.30.2 Конструктор(ы)

10.30.2.1 Velocity()

Velocity::Velocity (
const World2D & W_) [inline]

Конструктор

Аргументы

in	W↩ _	константная ссылка на решаемую задачу

См. определение в файле Velocity.h строка 109

```
109                                 :
110         W(W_)
111     {
112         virtualVortexesParams.resize(0);
113     };
```

10.30.2.2 ~Velocity()

virtual Velocity::~Velocity ( ) [inline], [virtual]

Деструктор

См. определение в файле Velocity.h строка 161

```
161 { };
```

10.30.3 Методы

10.30.3.1 CalcConvVelo()

void Velocity::CalcConvVelo ( )

Вычисление конвективных скоростей вихрей и виртуальных вихрей в вихревом следе

Вызывает 2 раза функцию CalcConvVeloToSetOfPoints

Предупреждения

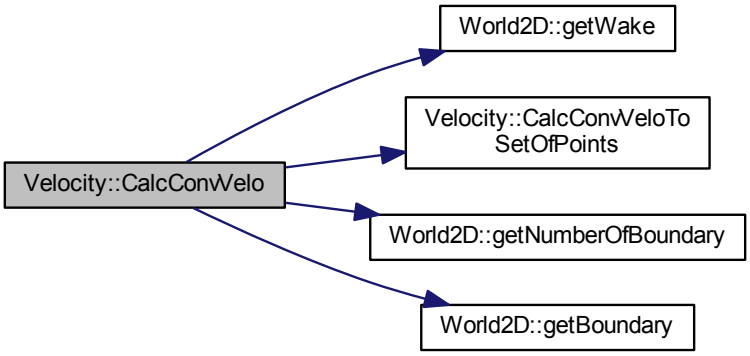
скорости приплюсовываются к уже имеющимся

См. определение в файле Velocity.cpp строка 44

```
45 {
46     /*
47     if (parallel.myidWork == 0)
48     {
49         std::ostringstream ss1;
50         ss1 << "wakeFile";
51         std::ofstream wakefile(ss1.str());
52         for (int i = 0; i < wake.vtx.size(); i++)
53             wakefile << wake.vtx[i].r()[0] << " " << wake.vtx[i].r()[1] << " " << wake.vtx[i].g() <<
    std::endl;
54         wakefile.close();
55     }
56     */
57
58 #if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_CONV_TOWAKE))
59     GPUCalcConvVeloToSetOfPoints(W.getWake(), wakeVortexesParams.
    convVelo, wakeVortexesParams.epsastWake);
60 #else
61     CalcConvVeloToSetOfPoints(W.getWake(),
    wakeVortexesParams.convVelo, wakeVortexesParams.
    epsastWake);
```

```
#endif

    std::vector<Point2D> nullVector(0);

    for (size_t bou = 0; bou < W.getNumberOfBoundary(); ++bou)
    {
#if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_CONV_TOBOU))
        GPUCalcConvVeloToSetOfPoints(W.getBoundary(bou).virtualWake, nullVector,
    virtualVortexesParams[bou].epsastWake, true);
#else
        CalcConvVeloToSetOfPoints(W.getBoundary(bou).
    virtualWake, nullVector, virtualVortexesParams[bou].epsastWake, true);
#endif
    }
}//CalcConvVelo()
```

Граф вызовов:



10.30.3.2 CalcDiffVelo()

void Velocity::CalcDiffVelo ( )

Вычисление диффузионных скоростей вихрей и виртуальных вихрей в вихревом следе

Вызывает 4 раза функцию CalcDiffVeloToSetOfPoints

Предупреждения

скорости приплюсовываются к уже имеющимся

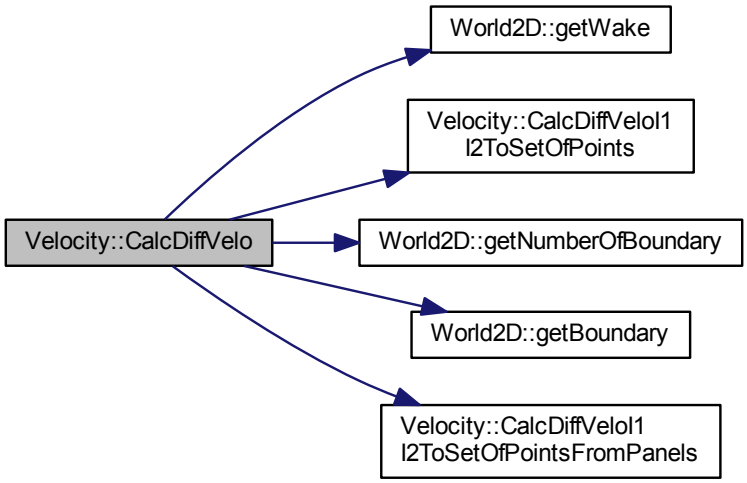
См. определение в файле Velocity.cpp строка 78

```
{
    double tTemp[4];

    tTemp[0] = omp_get_wtime();

    //пелена на пелену
#if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_I1I2))
    GPUCalcDiffVeloI1I2ToSetOfPoints(W.getWake(), wakeVortexesParams.
```

```
epsastWake, W.getWake(), wakeVortexesParams.
I1, wakeVortexesParams.I2, true);
#else
  CalcDiffVeloI1I2ToSetOfPoints(W.getWake(),
wakeVortexesParams.epsastWake, W.getWake(),
wakeVortexesParams.I1, wakeVortexesParams.
I2);
#endif


  tTemp[1] = omp_get_wtime();

  for (size_t bou = 0; bou < W.getNumberOfBoundary(); ++bou)
  {
     //не нужно, т.к. сделано выше перед началом вычисления скоростей
     //W.getNonConstBoundary(bou).virtualWake.WakeSynchronize();

     //виртуальные на границе на след
#if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_I1I2))
       GPUCalcDiffVeloI1I2ToSetOfPointsFromPanels(W.getWake(),
wakeVortexesParams.epsastWake, W.getBoundary(bou),
wakeVortexesParams.I1, wakeVortexesParams.
I2);
#else
       CalcDiffVeloI1I2ToSetOfPointsFromPanels(
W.getWake(), wakeVortexesParams.epsastWake,
W.getBoundary(bou), wakeVortexesParams.I1,
wakeVortexesParams.I2);
#endif

       // след на виртуальные
#if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_I1I2))
       GPUCalcDiffVeloI1I2ToSetOfPoints(W.getBoundary(bou).
virtualWake, virtualVortexesParams[bou].epsastWake,
W.getWake(), virtualVortexesParams[bou].I1,
virtualVortexesParams[bou].I2);
#else
       CalcDiffVeloI1I2ToSetOfPoints(W.
getBoundary(bou).virtualWake, virtualVortexesParams[bou].
epsastWake, W.getWake(), virtualVortexesParams[bou].I1,
virtualVortexesParams[bou].I2);
#endif

       for (size_t targetBou = 0; targetBou < W.getNumberOfBoundary(); ++targetBou)
       {
       // виртуальные на виртуальные
#if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_I1I2))
           GPUCalcDiffVeloI1I2ToSetOfPointsFromPanels(W.getBoundary(targetBou).
virtualWake, virtualVortexesParams[targetBou].epsastWake,
W.getBoundary(bou), virtualVortexesParams[targetBou].I1,
virtualVortexesParams[targetBou].I2);
#else
           CalcDiffVeloI1I2ToSetOfPointsFromPanels(
W.getBoundary(targetBou).virtualWake,
virtualVortexesParams[targetBou].epsastWake, W.getBoundary(bou),
virtualVortexesParams[targetBou].I1, virtualVortexesParams[
targetBou].I2);
#endif
       }
  } //for bou

tTemp[2] = omp_get_wtime();

  Point2D I2;
  double I1;

  for (size_t vt = 0; vt < wakeVortexesParams.diffVelo.size(); ++vt)
  {
     I2 = wakeVortexesParams.I2[vt];
     I1 = wakeVortexesParams.I1[vt];
     if (fabs(I1) < 1.e-8)
        wakeVortexesParams.diffVelo[vt] = { 0.0, 0.0 };
     else
        wakeVortexesParams.diffVelo[vt] = I2 * (1.0 / (I1 *
wakeVortexesParams.epsastWake[vt]));
  }

  for (size_t targetBou = 0; targetBou < W.getNumberOfBoundary(); ++targetBou)
  for (size_t vt = 0; vt < virtualVortexesParams[targetBou].diffVelo.size(); ++vt)
  {
     I2 = virtualVortexesParams[targetBou].I2[vt];
     I1 = virtualVortexesParams[targetBou].I1[vt];

     if (fabs(I1) < 1.e-8)
        virtualVortexesParams[targetBou].diffVelo[vt] = { 0.0, 0.0 };
     else
        virtualVortexesParams[targetBou].diffVelo[vt] = I2 * (1.0 / (I1 *
```

```
    virtualVortexesParams[targetBou].epsastWake[vt]));
149     }
150
151     tTemp[3] = omp_get_wtime();
152
153 #pragma warning (push)
154 #pragma warning (disable: 4010)
155     //W.getInfo('t') << "Times_diff:" << \
156         tTemp[1] - tTemp[0] << " " << \
157         tTemp[2] - tTemp[1] << " " << \
158         tTemp[3] - tTemp[2] << " " << \
159         std::endl;
160 #pragma warning (pop)
161
162 }//CalcDiffVelo()
```

Граф вызовов:

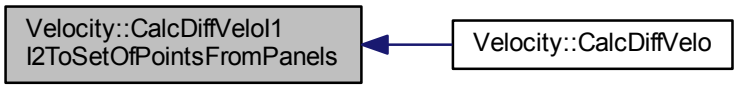


10.30.3.3 CalcDiffVeloI1I2ToSetOfPointsFromPanels()

```
virtual void Velocity::CalcDiffVeloI1I2ToSetOfPointsFromPanels (
            const WakeDataBase & pointsDb,
            const std::vector< double > & domainRadius,
            const Boundary & bnd,
            std::vector< double > & I1,
            std::vector< Point2D > & I2 )   [pure virtual]
```

Замещается в VelocityBiotSavart.

Граф вызова функции:



10.30.4 Данные класса

10.30.4.1 virtualVortexesParams

std::vector<VortexesParams> Velocity::virtualVortexesParams

Вектор струтур, определяющий параметры виртуальных вихрей для профилей

См. определение в файле Velocity.h строка 103

10.30.4.2 W

const World2D& Velocity::W [protected]

Константная ссылка на решаемую задачу

См. определение в файле Velocity.h строка 96

10.30.4.3 wakeVortexesParams

VortexesParams Velocity::wakeVortexesParams

Струтура, определяющая параметры вихрей в следе

См. определение в файле Velocity.h строка 100

Объявления и описания членов классов находятся в файлах:

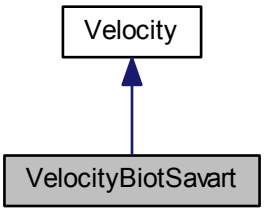
- Velocity/Velocity.h
- Velocity/Velocity.cpp

10.31 Класс VelocityBiotSavart

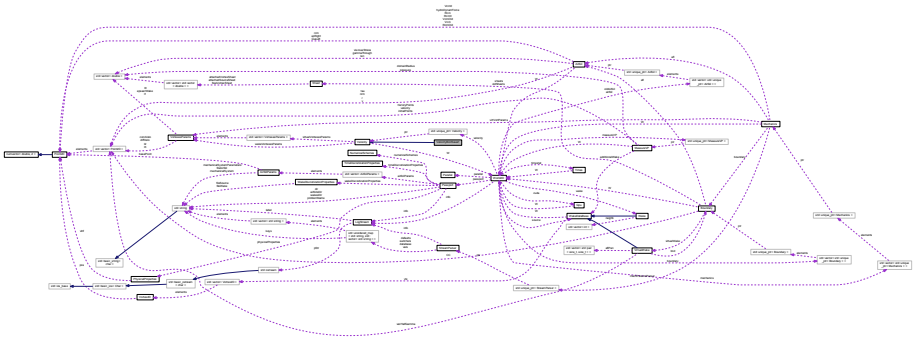
Класс, определяющий способ вычисления скоростей

#include <VelocityBiotSavart.h>

Граф наследования:VelocityBiotSavart:



Граф связей класса VelocityBiotSavart:



Открытые члены

- VelocityBiotSavart (const World2D &W_)

  Конструктор
- virtual ~VelocityBiotSavart ()

  Деструктор
- virtual void CalcConvVeloToSetOfPoints (const WakeDataBase &pointsDb, std::vector< Point2D > &velo, std::vector< double > &domainRadius, bool onlyRadius=false)

  Вычисление конвективных скоростей и радиусов вихревых доменов в заданном наборе точек от следа
- virtual void CalcDiffVeloI1I2ToSetOfPoints (const WakeDataBase &pointsDb, const std::vector< double > &domainRadius, const WakeDataBase &vorticesDb, std::vector< double > &I1, std↩ ::vector< Point2D > &I2)

  Вычисление числителей и знаменателей диффузионных скоростей в заданном наборе точек
- virtual void CalcDiffVeloI1I2ToSetOfPointsFromPanels (const WakeDataBase &pointsDb, const std::vector< double > &domainRadius, const Boundary &bnd, std::vector< double > &I1, std↩ ::vector< Point2D > &I2)

- void CalcConvVelo ()

  Вычисление конвективных скоростей вихрей и виртуальных вихрей в вихревом следе
- void CalcDiffVelo ()

  Вычисление диффузионных скоростей вихрей и виртуальных вихрей в вихревом следе

Открытые атрибуты

- VortexesParams wakeVortexesParams

  Струтура, определяющая параметры вихрей в следе
- std::vector< VortexesParams > virtualVortexesParams

  Вектор струтур, определяющий параметры виртуальных вихрей для профилей

Защищенные данные

- const World2D & W

  Константная ссылка на решаемую задачу

10.31.1 Подробное описание

Класс, определяющий способ вычисления скоростей

Способ вычисления скоростей

- напрямую по закону Био — Савара

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле VelocityBiotSavart.h строка 59

10.31.2 Конструктор(ы)

10.31.2.1 VelocityBiotSavart()

```
VelocityBiotSavart::VelocityBiotSavart (
            const World2D & W_ )
```

Конструктор

Аргументы

in	W↩ _	константная ссылка на решаемую задачу

См. определение в файле VelocityBiotSavart.cpp строка 45

```
45                                          :
46     Velocity(W_)
47 {
48 };
```

10.31.2.2 ~VelocityBiotSavart()

VelocityBiotSavart::~VelocityBiotSavart ( ) [virtual]

Деструктор

См. определение в файле VelocityBiotSavart.cpp строка 51

```
52 {
53 };
```

10.31.3 Методы

10.31.3.1 CalcConvVelo()

void Velocity::CalcConvVelo ( ) [inherited]

Вычисление конвективных скоростей вихрей и виртуальных вихрей в вихревом следе

Вызывает 2 раза функцию CalcConvVeloToSetOfPoints

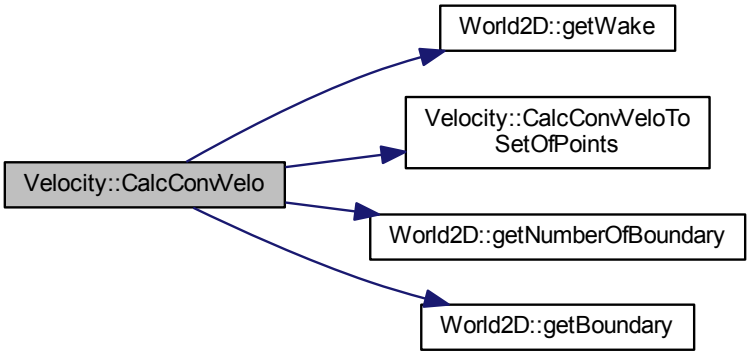
Предупреждения

скорости приплюсовываются к уже имеющимся

См. определение в файле Velocity.cpp строка 44

```
{
    /*
    if (parallel.myidWork == 0)
    {
        std::ostringstream ss1;
        ss1 << "wakeFile";
        std::ofstream wakefile(ss1.str());
        for (int i = 0; i < wake.vtx.size(); i++)
            wakefile << wake.vtx[i].r()[0] << " " << wake.vtx[i].r()[1] << " " << wake.vtx[i].g() << std::endl;
        wakefile.close();
    }
    */

#if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_CONV_TOWAKE))
    GPUCalcConvVeloToSetOfPoints(W.getWake(), wakeVortexesParams.convVelo, wakeVortexesParams.epsastWake);
#else
    CalcConvVeloToSetOfPoints(W.getWake(), wakeVortexesParams.convVelo, wakeVortexesParams.epsastWake);
#endif

    std::vector<Point2D> nullVector(0);

    for (size_t bou = 0; bou < W.getNumberOfBoundary(); ++bou)
    {
#if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_CONV_TOBOU))
        GPUCalcConvVeloToSetOfPoints(W.getBoundary(bou).virtualWake, nullVector, virtualVortexesParams[bou].epsastWake, true);
#else
        CalcConvVeloToSetOfPoints(W.getBoundary(bou).virtualWake, nullVector, virtualVortexesParams[bou].epsastWake, true);
#endif
    }
}//CalcConvVelo()
```

Граф вызовов:



10.31.3.2 CalcConvVeloToSetOfPoints()

```
void VelocityBiotSavart::CalcConvVeloToSetOfPoints (
            const WakeDataBase & pointsDb,
            std::vector< Point2D > & velo,
            std::vector< double > & domainRadius,
            bool onlyRadius = false )   [virtual]
```

Вычисление конвективных скоростей и радиусов вихревых доменов в заданном наборе точек от следа

Аргументы

in	pointsDb	константная ссылка на базу данных пелены из вихрей, в которых надо сосчитать конвективные скорости
out	velo	ссылка на вектор скоростей в требуемых точках
out	domainRadius	ссылка на вектор радиусов вихревых доменов
in	onlyRadius	признак вычисления только радиусов доменом, иначе и скоростей тоже (по умолчанию false)

Предупреждения

Использует OMP, MPI

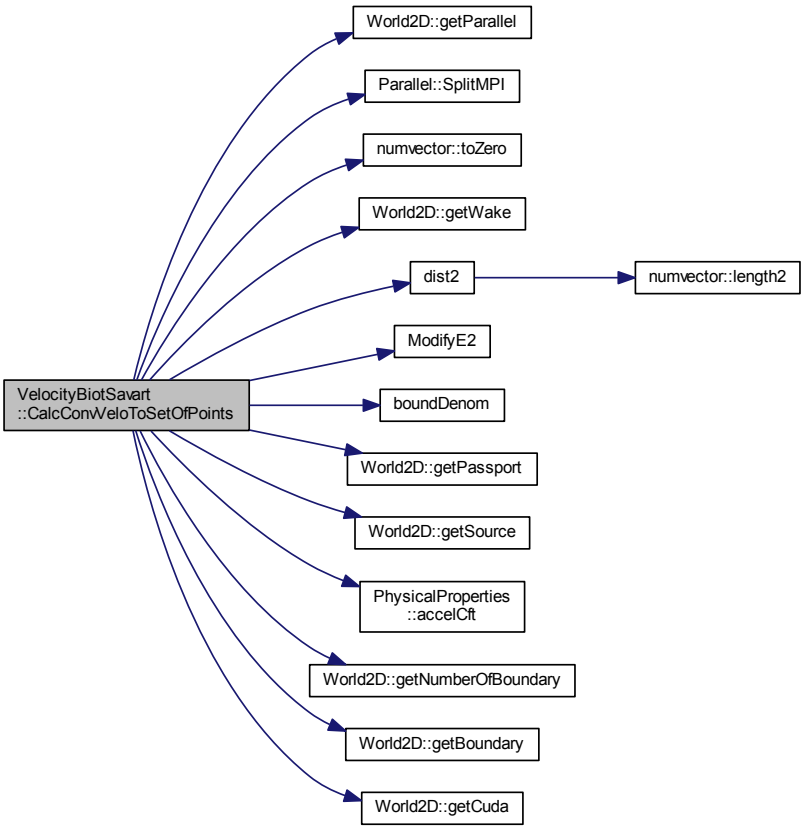
Замещает Velocity.

См. определение в файле VelocityBiotSavart.cpp строка 80

```
81 {
82     double tCPUSTART, tCPUEND;
83
84     tCPUSTART = omp_get_wtime();
85     std::vector<Point2D> selfVelo;
86
87     const int& id = W.getParallel().myidWork;
88
89     parProp par = W.getParallel().SplitMPI(pointsDb.
    vtx.size(), true);
90
91     std::vector<Vortex2D> locPoints;
92     locPoints.resize(par.myLen);
93
94     //Заполнение "своей" части массива скоростей
95     MPI_Scatterv(const_cast<std::vector<Vortex2D>&>(pointsDb.vtx).data(), par.
    len.data(), par.disp.data(), Vortex2D::mpiVortex2D, \
96                  locPoints.data(), par.myLen, Vortex2D::mpiVortex2D, 0,
    W.getParallel().commWork);
97
98     double cft = IDPI;
99
100     std::vector<Point2D> locConvVelo;
101     if (!onlyRadius)
102         locConvVelo.resize(par.myLen);
103
104     std::vector<double> locDomRadius;
105     locDomRadius.resize(par.myLen);
106
107 #pragma warning (push)
108 #pragma warning (disable: 4101)
109     //Локальные переменные для цикла
110     Point2D velI;
111     Point2D tempVel;
112     double dst2eps, dst2;
113 #pragma warning (pop)
114
115 #pragma omp parallel for default(none) shared(locConvVelo, locDomRadius, locPoints, cft, par, onlyRadius)
     private(tempVel, velI, dst2, dst2eps) schedule(dynamic, DYN_SCHEDULE)
116     for (int i = 0; i < par.myLen; ++i)
117     {
118         double ee2[3] = { 10000.0, 10000.0, 10000.0 };
119
120         velI.toZero();
121
122         const Point2D& posI = locPoints[i].r();
123
124         for (size_t j = 0; j < W.getWake().vtx.size(); ++j)
125         {
126             const Point2D& posJ = W.getWake().vtx[j].r();
127
128
129             dst2 = dist2(posI, posJ);
130
131             //Модифицируем массив квадратов расстояний до ближайших вихрей из wake
132             ModifyE2(ee2, dst2);
133
```

```
            if (!onlyRadius)
            {
                const double& gamJ = W.getWake().vtx[j].g();

                tempVel.toZero();
                dst2eps = boundDenom(dst2, W.getPassport().wakeDiscretizationProperties.eps2); //Сглаживать надо!!!

                tempVel = { -posI[1] + posJ[1], posI[0] - posJ[0] };
                tempVel *= (gamJ / dst2eps);
                velI += tempVel;
            }
        }

        if (!onlyRadius)
            for (size_t j = 0; j < W.getSource().vtx.size(); ++j)
            {
                const Point2D& posJ = W.getSource().vtx[j].r();
                const double& gamJ = W.getPassport().physicalProperties.accelCft() * W.getSource().vtx[j].g();

                tempVel.toZero();

                dst2 = dist2(posI, posJ);
                dst2eps = boundDenom(dst2, W.getPassport().wakeDiscretizationProperties.eps2); //Сглаживать надо!!!

                tempVel = { posI[0] - posJ[0], posI[1] - posJ[1] };
                tempVel *= (gamJ / dst2eps);
                velI += tempVel;
            }

        for (size_t s = 0; s < W.getNumberOfBoundary(); ++s)
        {
            for (size_t j = 0; j < W.getBoundary(s).virtualWake.vtx.size(); ++j)
            {
                const Point2D& posJ = W.getBoundary(s).virtualWake.vtx[j].r();
                dst2 = dist2(posI, posJ);

                //Модифицируем массив квадратов расстояний до ближайших вихрей из virtualWake
                ModifyE2(ee2, dst2);
            }
        }

        if (!onlyRadius)
        {
            velI *= cft;
            locConvVelo[i] = velI;
        }

        locDomRadius[i] = std::max(sqrt((ee2[0] + ee2[1] + ee2[2]) / 3.0), 2.0*W.getPassport().wakeDiscretizationProperties.epscol);
    }


    if (!onlyRadius)
    {
        if (id == 0)
            selfVelo.resize(pointsDb.vtx.size());
        MPI_Gatherv(locConvVelo.data(), par.myLen, Point2D::mpiPoint2D, selfVelo.data(), par.len.data(), par.disp.data(), Point2D::mpiPoint2D, 0, W.getParallel().commWork);
    }

    //if (id == 0)
    //это нужно всем, т.к. ниже стоит Allgatherv
    domainRadius.resize(par.totalLen);

    MPI_Allgatherv(locDomRadius.data(), par.myLen, MPI_DOUBLE, domainRadius.data(), par.len.data(), par.disp.data(), MPI_DOUBLE, W.getParallel().commWork);

    if ( (!onlyRadius) && (id == 0) )
        for (size_t i = 0; i < velo.size(); ++i)
            velo[i] += selfVelo[i];

    tCPUEND = omp_get_wtime();
    //W.getInfo('t') << "CONV_CPU: " << tCPUEND - tCPUSTART << std::endl;

}//CalcConvVeloToSetOfPoints(...)
```

Граф вызовов:



10.31.3.3 CalcDiffVelo()

void Velocity::CalcDiffVelo ( ) [inherited]

Вычисление диффузионных скоростей вихрей и виртуальных вихрей в вихревом следе

Вызывает 4 раза функцию CalcDiffVeloToSetOfPoints

Предупреждения

скорости приплюсовываются к уже имеющимся

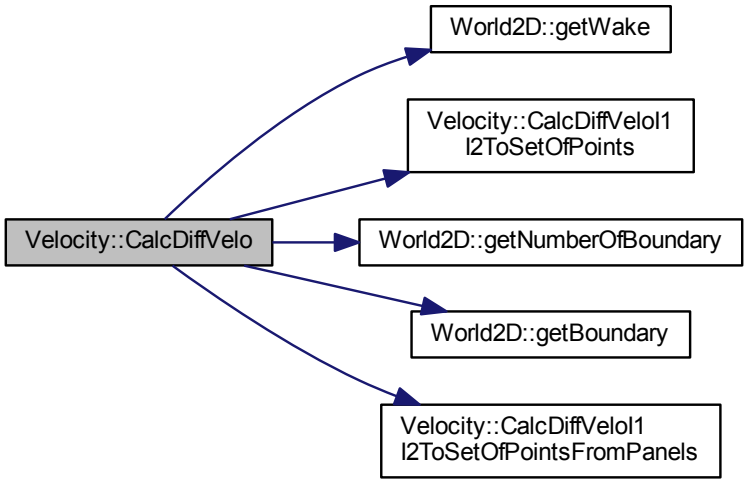
См. определение в файле Velocity.cpp строка 78

```
79 {
80     double tTemp[4];
81
82     tTemp[0] = omp_get_wtime();
83
84     //пелена на пелену
85 #if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_I1I2))
```

```
86      GPUCalcDiffVeloI1I2ToSetOfPoints(W.getWake(), wakeVortexesParams.
    epsastWake, W.getWake(), wakeVortexesParams.
    I1, wakeVortexesParams.I2, true);
87  #else
88      CalcDiffVeloI1I2ToSetOfPoints(W.getWake(),
    wakeVortexesParams.epsastWake, W.getWake(),
    wakeVortexesParams.I1, wakeVortexesParams.
    I2);
89  #endif
90
91
92      tTemp[1] = omp_get_wtime();
93
94      for (size_t bou = 0; bou < W.getNumberOfBoundary(); ++bou)
95      {
96          //не нужно, т.к. сделано выше перед началом вычисления скоростей
97          //W.getNonConstBoundary(bou).virtualWake.WakeSynchronize();
98
99          //виртуальные на границе на след
100 #if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_I1I2))
101         GPUCalcDiffVeloI1I2ToSetOfPointsFromPanels(W.getWake(),
    wakeVortexesParams.epsastWake, W.getBoundary(bou),
    wakeVortexesParams.I1, wakeVortexesParams.
    I2);
102 #else
103         CalcDiffVeloI1I2ToSetOfPointsFromPanels(
    W.getWake(), wakeVortexesParams.epsastWake,
    W.getBoundary(bou), wakeVortexesParams.I1,
    wakeVortexesParams.I2);
104 #endif
105
106         // след на виртуальные
107 #if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_I1I2))
108         GPUCalcDiffVeloI1I2ToSetOfPoints(W.getBoundary(bou).
    virtualWake, virtualVortexesParams[bou].epsastWake,
    W.getWake(), virtualVortexesParams[bou].I1,
    virtualVortexesParams[bou].I2);
109 #else
110         CalcDiffVeloI1I2ToSetOfPoints(W.
    getBoundary(bou).virtualWake, virtualVortexesParams[bou].
    epsastWake, W.getWake(), virtualVortexesParams[bou].I1,
    virtualVortexesParams[bou].I2);
111 #endif
112
113         for (size_t targetBou = 0; targetBou < W.getNumberOfBoundary(); ++targetBou)
114         {
115         // виртуальные на виртуальные
116 #if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_I1I2))
117             GPUCalcDiffVeloI1I2ToSetOfPointsFromPanels(W.getBoundary(targetBou).
    virtualWake, virtualVortexesParams[targetBou].epsastWake,
    W.getBoundary(bou), virtualVortexesParams[targetBou].I1,
    virtualVortexesParams[targetBou].I2);
118 #else
119             CalcDiffVeloI1I2ToSetOfPointsFromPanels(
    W.getBoundary(targetBou).virtualWake,
    virtualVortexesParams[targetBou].epsastWake, W.getBoundary(bou),
    virtualVortexesParams[targetBou].I1, virtualVortexesParams[
    targetBou].I2);
120 #endif
121         }
122     } //for bou
123
124 tTemp[2] = omp_get_wtime();
125
126     Point2D I2;
127     double I1;
128
129     for (size_t vt = 0; vt < wakeVortexesParams.diffVelo.size(); ++vt)
130     {
131         I2 = wakeVortexesParams.I2[vt];
132         I1 = wakeVortexesParams.I1[vt];
133         if (fabs(I1) < 1.e-8)
134             wakeVortexesParams.diffVelo[vt] = { 0.0, 0.0 };
135         else
136             wakeVortexesParams.diffVelo[vt] = I2 * (1.0 / (I1 *
    wakeVortexesParams.epsastWake[vt]));
137     }
138
139     for (size_t targetBou = 0; targetBou < W.getNumberOfBoundary(); ++targetBou)
140     for (size_t vt = 0; vt < virtualVortexesParams[targetBou].diffVelo.size(); ++vt)
141     {
142         I2 = virtualVortexesParams[targetBou].I2[vt];
143         I1 = virtualVortexesParams[targetBou].I1[vt];
144
145         if (fabs(I1) < 1.e-8)
146             virtualVortexesParams[targetBou].diffVelo[vt] = { 0.0, 0.0 };
147         else
```

```
148            virtualVortexesParams[targetBou].diffVelo[vt] = I2 * (1.0 / (I1 *
    virtualVortexesParams[targetBou].epsastWake[vt]));
149     }
150
151     tTemp[3] = omp_get_wtime();
152
153 #pragma warning (push)
154 #pragma warning (disable: 4010)
155     //W.getInfo('t') << "Times_diff:" << \
156         tTemp[1] - tTemp[0] << " " << \
157         tTemp[2] - tTemp[1] << " " << \
158         tTemp[3] - tTemp[2] << " " << \
159         std::endl;
160 #pragma warning (pop)
161
162 }//CalcDiffVelo()
```

Граф вызовов:



10.31.3.4 CalcDiffVeloI1I2ToSetOfPoints()

```
void VelocityBiotSavart::CalcDiffVeloI1I2ToSetOfPoints (
            const WakeDataBase & pointsDb,
            const std::vector< double > & domainRadius,
            const WakeDataBase & vorticesDb,
            std::vector< double > & I1,
            std::vector< Point2D > & I2 )   [virtual]
```

Вычисление числителей и знаменателей диффузионных скоростей в заданном наборе точек

Аргументы

in	pointsDb	константная ссылка на базу данных пелены из вихрей, в которых надо сосчитать диффузионные скорости
in	domainRadius	константная ссылка на вектор радиусов вихревых доменов

Аргументы

in	vorticesDb	константная ссылка на на базу данных пелены из вихрей,от которых надо сосчитать влияния на points
out	I1	ссылка на вектор величин I1 (знаменателей в диффузионных скоростях) в требуемых точках
out	I2	ссылка на вектор величин I2 (числителей в диффузионных скоростях) в требуемых точках

Предупреждения

Использует OMP, MPI

Необходимо сделать Понять природу магической константы 8.0 и синхронизировать с GPU

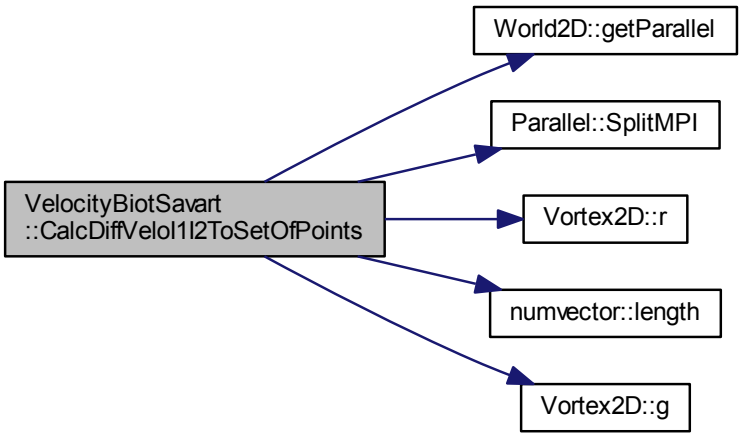
Замещает Velocity.

См. определение в файле VelocityBiotSavart.cpp строка 278

```
279 {
280     double tCPUSTART, tCPUEND;
281
282     tCPUSTART = omp_get_wtime();
283
284     std::vector<double> selfI1;
285     std::vector<Point2D> selfI2;
286
287     const int& id = W.getParallel().myidWork;
288
289     parProp par = W.getParallel().SplitMPI(pointsDb.
    vtx.size());
290
291     std::vector<Vortex2D> locPoints;
292     locPoints.resize(par.myLen);
293
294     MPI_Scatterv(const_cast<std::vector<Vortex2D>&>(pointsDb.vtx).data(), par.
    len.data(), par.disp.data(), Vortex2D::mpiVortex2D, \
295         locPoints.data(), par.myLen, Vortex2D::mpiVortex2D, 0,
    W.getParallel().commWork);
296
297     std::vector<double> locI1(par.myLen, 0.0);
298     std::vector<Point2D> locI2(par.myLen, {0.0, 0.0});
299
300 #pragma warning (push)
301 #pragma warning (disable: 4101)
302     //Локальные переменные для цикла
303     Point2D Rij;
304     double rij, expr;
305     double diffRadius;
306     double left;
307     double right;
308     double posJx;
309 #pragma warning (pop)
310
311 #pragma omp parallel for default(none) shared(locI1, locI2, domainRadius, locPoints, vorticesDb, par)
     private(Rij, rij, expr, diffRadius, left, right, posJx)
312     for (int i = 0; i < par.myLen; ++i)
313     {
314         const Vortex2D& vtxI = locPoints[i];
315
317
318         diffRadius =  8.0 * domainRadius[i + par.myDisp];
319
320         left = vtxI.r()[0] - diffRadius;
321         right = vtxI.r()[0] + diffRadius;
322
323         for (size_t j = 0; j < vorticesDb.vtx.size(); ++j)
324         {
325             const Vortex2D& vtxJ = vorticesDb.vtx[j];
326             posJx = vtxJ.r()[0];
327
328             if ((left < posJx) && (posJx < right))
```

```
            {
                Rij = vtxI.r() - vtxJ.r();
                rij = Rij.length();
                if (rij < diffRadius && rij > 1.e-10)
                {
                    expr = exp(-rij / domainRadius[i + par.myDisp]);
                    locI2[i] += (vtxJ.g()* expr / rij) * Rij;
                    locI1[i] += vtxJ.g()*expr;
                }
            }//if (rij>1e-6)
        }//for j
    } // for r

    //std::ostringstream sss;
    //sss << "velo_";
    //std::ofstream veloFile(sss.str());
    //for (int i = 0; i < domainRadius.size(); ++i)
    //  veloFile << domainRadius[i] << std::endl;
    //veloFile.close();

    if (id == 0)
    {
        selfI1.resize(pointsDb.vtx.size(), 0.0);
        selfI2.resize(pointsDb.vtx.size(), { 0.0, 0.0 });
    }

    MPI_Gatherv(locI1.data(), par.myLen, MPI_DOUBLE,          selfI1.data(), par.
len.data(), par.disp.data(), MPI_DOUBLE,          0, W.getParallel().
commWork);

    MPI_Gatherv(locI2.data(), par.myLen, Point2D::mpiPoint2D, selfI2.data(), par.
len.data(), par.disp.data(), Point2D::mpiPoint2D, 0, W.
getParallel().commWork);

    if (id == 0)
    for (size_t i = 0; i < I1.size(); ++i)
    {
        I1[i] += selfI1[i];
        I2[i] += selfI2[i];
    }

    tCPUEND = omp_get_wtime();
    //W.getInfo('t') << "DIFF_CPU: " << tCPUEND - tCPUSTART << std::endl;
}//CalcDiffVeloI1I2ToSetOfPoints(...)
```

Граф вызовов:



10.31.3.5 CalcDiffVeloI1I2ToSetOfPointsFromPanels()

```
void VelocityBiotSavart::CalcDiffVeloI1I2ToSetOfPointsFromPanels (
            const WakeDataBase & pointsDb,
            const std::vector< double > & domainRadius,
            const Boundary & bnd,
            std::vector< double > & I1,
            std::vector< Point2D > & I2 )   [virtual]
```

Необходимо сделать Понять природу магической константы 8.0 и синхронизировать с GPU

Необходимо сделать Сделать переменной и синхронизировать с GPU

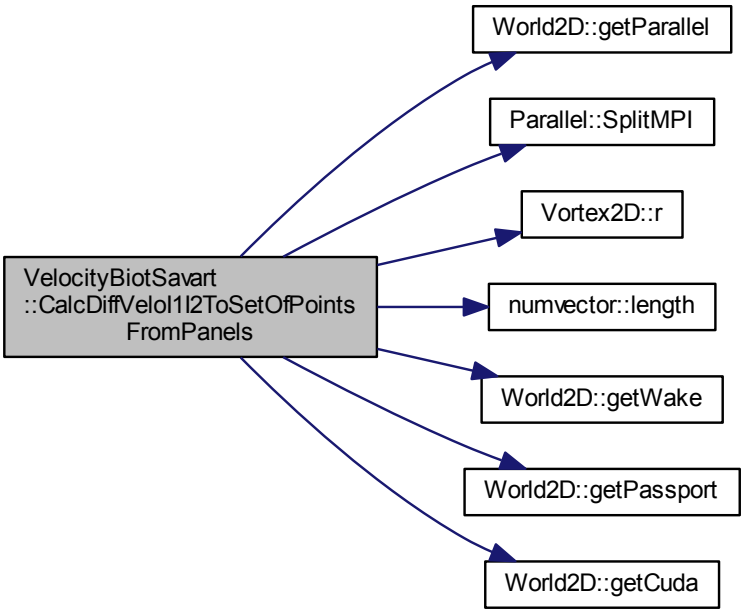
Замещает Velocity.

См. определение в файле VelocityBiotSavart.cpp строка 373

```
374 {
375     double tCPUSTART, tCPUEND;
376
377     tCPUSTART = omp_get_wtime();
378
379     std::vector<double> selfI1;
380     std::vector<Point2D> selfI2;
381
382     const int& id = W.getParallel().myidWork;
383     parProp par = W.getParallel().SplitMPI(pointsDb.
    vtx.size());
384
385
386     //синхронизация свободного вихревого слоя
387     std::vector<double> freeVortexSheetGamma;
388     int sz;
389
390     if (id == 0)
391     {
392         sz = (int)(bnd.sheets.freeVortexSheet.size());
393         freeVortexSheetGamma.resize(sz, 0.0);
394         for (size_t j = 0; j < bnd.sheets.freeVortexSheet.size(); ++j)
395             freeVortexSheetGamma[j] = bnd.sheets.freeVortexSheet[j][0];
396     }
397     MPI_Bcast(&sz, 1, MPI_INT, 0, W.getParallel().commWork);
398     if (id != 0)
399         freeVortexSheetGamma.resize(sz, 0.0);
400
401     MPI_Bcast(freeVortexSheetGamma.data(), (int)sz, MPI_DOUBLE, 0, W.
    getParallel().commWork);
402
403
404     std::vector<Vortex2D> locPoints;
405     locPoints.resize(par.myLen);
406
407     MPI_Scatterv(const_cast<std::vector<Vortex2D>&>(pointsDb.vtx).data(), par.
    len.data(), par.disp.data(), Vortex2D::mpiVortex2D, \
408         locPoints.data(), par.myLen, Vortex2D::mpiVortex2D, 0,
    W.getParallel().commWork);
409
410     std::vector<double> locI1(par.myLen, 0.0);
411     std::vector<Point2D> locI2(par.myLen, {0.0, 0.0});
412
413 #pragma warning (push)
414 #pragma warning (disable: 4101)
415     //Локальные переменные для цикла
416     Point2D Rij;
417     double rij, expr;
418     double diffRadius;
419     double left;
420     double right;
421     double posJx;
422 #pragma warning (pop)
423
424
425 #pragma omp parallel for default(none) shared(locI1, locI2, domainRadius, locPoints, bnd, par, std::cout,
     freeVortexSheetGamma) private(Rij, rij, expr, diffRadius, left, right, posJx)
```

```
    for (int i = 0; i < par.myLen; ++i)
    {
        const Vortex2D& vtxI = locPoints[i];

        diffRadius = 8.0 * domainRadius[i + par.myDisp];

        left = vtxI.r()[0] - diffRadius;
        right = vtxI.r()[0] + diffRadius;

        for (size_t j = 0; j < bnd.sheets.freeVortexSheet.size(); ++j)
        {
            const int nQuadPt = 1;

            const double ptG = freeVortexSheetGamma[j] * bnd.afl.len[j] / nQuadPt;

            for (int q = 0; q < nQuadPt; ++q)
            {
                const Point2D& ptJ = bnd.CC[j] + bnd.afl.tau[j] * (q + 0.5) * bnd.afl.len[j] * (1.0 / nQuadPt);  // vorticesDb.vtx[j];
                posJx = ptJ[0];

                if ((left < posJx) && (posJx < right))
                {
                    Rij = vtxI.r() - ptJ;
                    rij = Rij.length();
                    if (rij < diffRadius && rij > 1.e-10)
                    {
                        expr = exp(-rij / domainRadius[i + par.myDisp]);
                        locI2[i] += (ptG * expr / rij) * Rij;
                        locI1[i] += ptG * expr;
                    }
                }//if (rij>1e-6)
            }
        }//for j
    } // for r

    //std::ostringstream sss;
    //sss << "velo_";
    //std::ofstream veloFile(sss.str());
    //for (int i = 0; i < domainRadius.size(); ++i)
    //  veloFile << domainRadius[i] << std::endl;
    //veloFile.close();

    if (id == 0)
    {
        selfI1.resize(pointsDb.vtx.size(), 0.0);
        selfI2.resize(pointsDb.vtx.size(), { 0.0, 0.0 });
    }

    MPI_Gatherv(locI1.data(), par.myLen, MPI_DOUBLE, selfI1.data(), par.len.data(), par.disp.data(), MPI_DOUBLE, 0, W.getParallel().commWork);
    MPI_Gatherv(locI2.data(), par.myLen, Point2D::mpiPoint2D, selfI2.data(), par.len.data(), par.disp.data(), Point2D::mpiPoint2D, 0, W.getParallel().commWork);


    if (id == 0)
    for (size_t i = 0; i < I1.size(); ++i)
    {
        I1[i] += selfI1[i];
        I2[i] += selfI2[i];
    }

    tCPUEND = omp_get_wtime();
    //W.getInfo('t') << "DIFF_CPU: " << tCPUEND - tCPUSTART << std::endl;
}//CalcDiffVeloI1I2ToSetOfPointsFromPanels(...)
```

Граф вызовов:



10.31.4 Данные класса

10.31.4.1 virtualVortexesParams

std::vector<VortexesParams> Velocity::virtualVortexesParams [inherited]

Вектор струтур, определяющий параметры виртуальных вихрей для профилей

См. определение в файле Velocity.h строка 103

10.31.4.2 W

const World2D& Velocity::W [protected], [inherited]

Константная ссылка на решаемую задачу

См. определение в файле Velocity.h строка 96

10.31.4.3 wakeVortexesParams

VortexesParams Velocity::wakeVortexesParams [inherited]

Струтура, определяющая параметры вихрей в следе

См. определение в файле Velocity.h строка 100

Объявления и описания членов классов находятся в файлах:

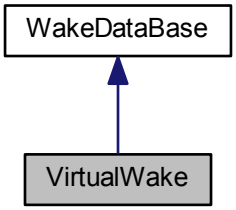
- Velocity/VelocityBiotSavart.h
- Velocity/VelocityBiotSavart.cpp

10.32 Класс VirtualWake

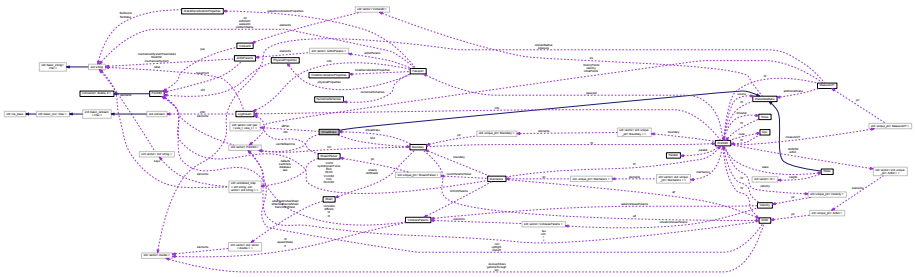
Класс, опеделяющий вихревой след (пелену)

#include <VirtualWake.h>

Граф наследования:VirtualWake:



Граф связей класса VirtualWake:



Открытые члены

- VirtualWake (const World2D &W_, const Boundary &bnd_)

  Конструктор инициализации
- ~VirtualWake ()

  Деструктор
- void ReadFromFile (const std::string &dir, const std::string &fileName)

  Считывание вихревого следа из файла
- void WakeSynchronize ()

  MPI-синхронизация вихревого следа

Открытые атрибуты

- const Boundary & bnd

  Константная ссылка на границу, которой принадлежит данный виртуальный след
- std::vector< Point2D > vecHalfGamma

  Скорость вихрей виртуального следа конкретного профиля (равна Gamma/2) используется для расчета давления
- std::vector< std::pair< size_t, size_t > > aflPan

  Пара чисел: номер профиля и номер панели, на которой рожден виртуальный вихрь
- const World2D & W

  Константная ссылка на решаемую задачу
- std::vector< Vortex2D > vtx

  Список вихревых элементов

10.32.1 Подробное описание

Класс, опеделяющий вихревой след (пелену)

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле VirtualWake.h строка 61

10.32.2 Конструктор(ы)

10.32.2.1 VirtualWake()

```
VirtualWake::VirtualWake (
            const World2D & W_,
            const Boundary & bnd_ )   [inline]
```

Конструктор инициализации

Аргументы

in	W↩ _	константная ссылка на решаемую задачу
in	bnd↩ _	константная ссылка на границу

См. определение в файле VirtualWake.h строка 80

```
81        : WakeDataBase(W_), bnd(bnd_) { };
```

10.32.2.2 ~VirtualWake()

VirtualWake::~VirtualWake ( ) [inline]

Деструктор

См. определение в файле VirtualWake.h строка 84

```
84 { };
```

10.32.3 Методы

10.32.3.1 ReadFromFile()

void WakeDataBase::ReadFromFile (
const std::string & dir,
const std::string & fileName) [inherited]

Считывание вихревого следа из файла

Аргументы

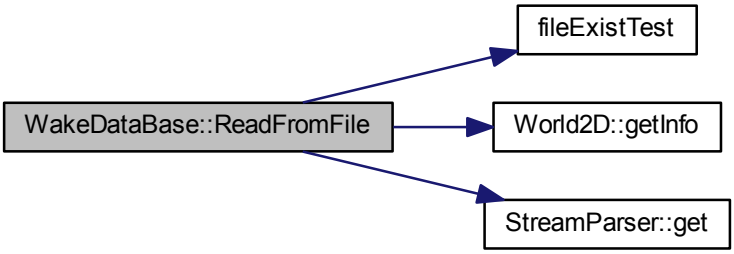
in	dir	константная ссылка на строку, задающую каталог, где лежит файл с вихревым следом
in	fileName	константная ссылка на строку, задающую имя файла с вихревым следом

См. определение в файле WakeDataBase.cpp строка 53

```
54 {
55     std::string filename = dir + fileName;
56     std::ifstream wakeFile;
57
58     if (fileExistTest(filename, W.getInfo()))
```

```
59    {
60        std::stringstream wakeFile(Preprocessor(filename).resultString);
61
62        LogStream XXX;
63        StreamParser wakeParser(XXX, "vortex wake file parser", wakeFile);
64
65        wakeParser.get("vtx", vtx);
66    }
67 }//ReadFromFile(...)
```

Граф вызовов:



Граф вызова функции:

10.32.4 Данные класса

10.32.4.1 aflPan

std::vector<std::pair<size_t, size_t> > VirtualWake::aflPan

Пара чисел: номер профиля и номер панели, на которой рожден виртуальный вихрь

См. определение в файле VirtualWake.h строка 74

10.32.4.2 bnd

const Boundary& VirtualWake::bnd

Константная ссылка на границу, которой принадлежит данный виртуальный след

См. определение в файле VirtualWake.h строка 68

10.32.4.3 vecHalfGamma

std::vector<Point2D> VirtualWake::vecHalfGamma

Скорость вихрей виртуального следа конкретного профиля (равна Gamma/2) используется для расчета давления

См. определение в файле VirtualWake.h строка 71

10.32.4.4 vtx

std::vector<Vortex2D> WakeDataBase::vtx [inherited]

Список вихревых элементов

См. определение в файле WakeDataBase.h строка 61

10.32.4.5 W

const World2D& WakeDataBase::W [inherited]

Константная ссылка на решаемую задачу

См. определение в файле WakeDataBase.h строка 55

Объявления и описания членов класса находятся в файле:

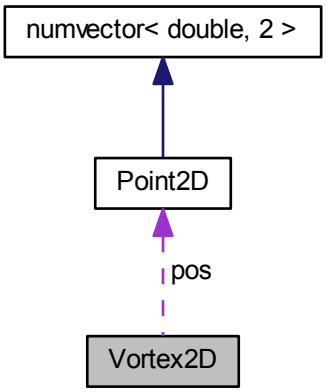
- VirtualWake/VirtualWake.h

10.33 Класс Vortex2D

Класс, опеделяющий двумерный вихревой элемент

#include <Vortex2D.h>

Граф связей класса Vortex2D:



Открытые члены

- Vortex2D ()

  Пустой конструктор
- Vortex2D (const Point2D &_r, const double _g)

  Конструктор инициализации
- ~Vortex2D ()

  Деструктор
- Point2D & r ()

  Функция для доступа к радиус-вектору вихря
- const Point2D & r () const

  Функция для доступа для чтения к радиус-вектору вихря
- double & g ()

  Функция для доступа к циркуляции вихря
- const double & g () const

  Функция для доступа для чтения к циркуляции вихря

Открытые статические члены

- static void CreateMpiType ()

  Создание MPI-описателя типа

Статические открытые данные

- static MPI_Datatype mpiVortex2D

  MPI-описатель типа
- static size_t offsPos
- static size_t offsGam

Закрытые данные

- Point2D pos

  Радиус-вектор вихря
- double gam

  Циркуляция вихря

10.33.1 Подробное описание

Класс, опеделяющий двумерный вихревой элемент

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Vortex2D.h строка 55

10.33.2 Конструктор(ы)

10.33.2.1 Vortex2D() [1/2]

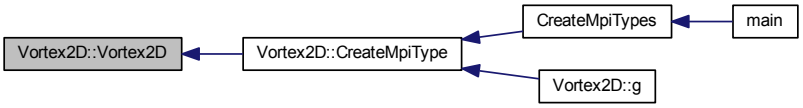
Vortex2D::Vortex2D ( ) [inline]

Пустой конструктор

См. определение в файле Vortex2D.h строка 72

```
72 {};
```

Граф вызова функции:



10.33.2.2 Vortex2D() [2/2]

```
Vortex2D::Vortex2D (
            const Point2D & _r,
            const double _g )   [inline]
```

Конструктор инициализации

Аргументы

in	↩ _↩ r	константная ссылка на радиус-вектор положения вихря
in	↩ _↩ g	циркуляция (интенсивность) вихря

См. определение в файле Vortex2D.h строка 78

```
79        : pos(_r), gam(_g)    { }
```

10.33.2.3 ~Vortex2D()

```
Vortex2D::~Vortex2D ( )   [inline]
```

Деструктор

См. определение в файле Vortex2D.h строка 82

```
82 {};
```

10.33.3 Методы

10.33.3.1 CreateMpiType()

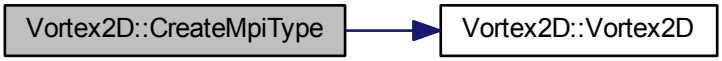
```
void Vortex2D::CreateMpiType ( )   [static]
```

Создание MPI-описателя типа

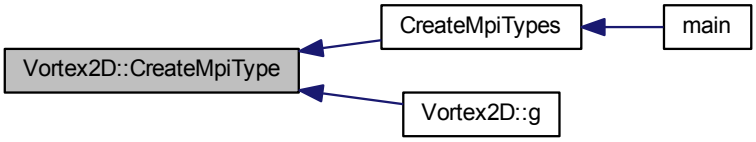
См. определение в файле Vortex2D.cpp строка 49

```
{
    int          len[4] = { 1, 2, 1, 1 };
    MPI_Aint     pos[4] = { 0, offsetof(Vortex2D, pos), offsetof(
Vortex2D, gam), sizeof(Vortex2D) };
    MPI_Datatype typ[4] = { MPI_LB, MPI_DOUBLE, MPI_DOUBLE, MPI_UB };

    MPI_Type_create_struct(4, len, pos, typ, &mpiVortex2D);
    MPI_Type_commit(&mpiVortex2D);

    offsPos = offsetof(Vortex2D, pos);
    offsGam = offsetof(Vortex2D, gam);
}
```

Граф вызовов:



Граф вызова функции:



10.33.3.2 g() [1/2]

double& Vortex2D::g ( ) [inline]

Функция для доступа к циркуляции вихря

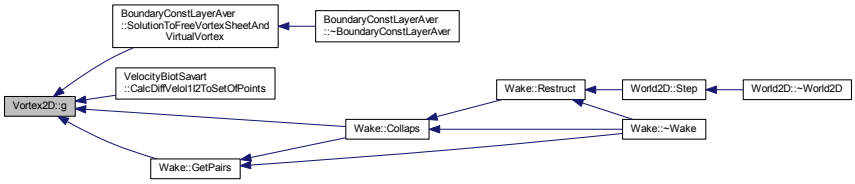
Возвращает

ссылка на циркуляцию вихря

См. определение в файле Vortex2D.h строка 94

```
{ return gam; }
```

Граф вызова функции:



10.33.3.3 g() [2/2]

const double& Vortex2D::g ( ) const [inline]

Функция для доступа для чтения к циркуляции вихря

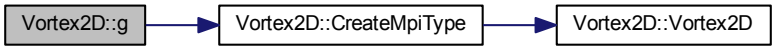
Возвращает

константная ссылка на циркуляцию вихря

См. определение в файле Vortex2D.h строка 98

```
98 { return gam; }
```

Граф вызовов:



10.33.3.4 r() [1/2]

Point2D& Vortex2D::r ( ) [inline]

Функция для доступа к радиус-вектору вихря

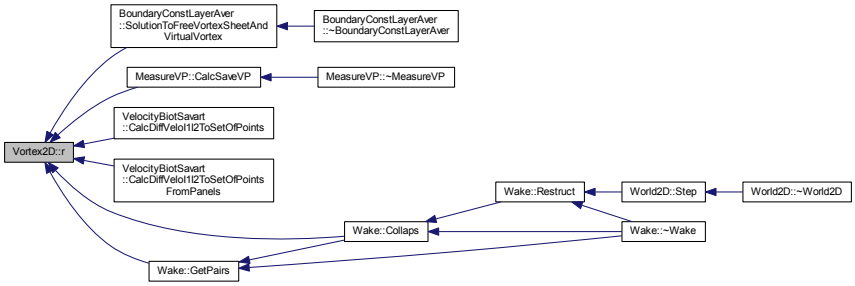
Возвращает

ссылка на радиус-вектор вихря

См. определение в файле Vortex2D.h строка 86

```
86 { return pos; }
```

Граф вызова функции:



10.33.3.5 r() [2/2]

const Point2D& Vortex2D::r ( ) const [inline]

Функция для доступа для чтения к радиус-вектору вихря

Возвращает

константная ссылка на радиус-вектор вихря

См. определение в файле Vortex2D.h строка 90

```
90 { return pos; }
```

10.33.4 Данные класса

10.33.4.1 gam

double Vortex2D::gam [private]

Циркуляция вихря

См. определение в файле Vortex2D.h строка 62

10.33.4.2 mpiVortex2D

MPI_Datatype Vortex2D::mpiVortex2D [static]

MPI-описатель типа

См. определение в файле Vortex2D.h строка 66

10.33.4.3 offsGam

size_t Vortex2D::offsGam [static]

См. определение в файле Vortex2D.h строка 69

10.33.4.4 offsPos

size_t Vortex2D::offsPos [static]

См. определение в файле Vortex2D.h строка 68

10.33.4.5 pos

Point2D Vortex2D::pos [private]

Радиус-вектор вихря

См. определение в файле Vortex2D.h строка 59

Объявления и описания членов классов находятся в файлах:

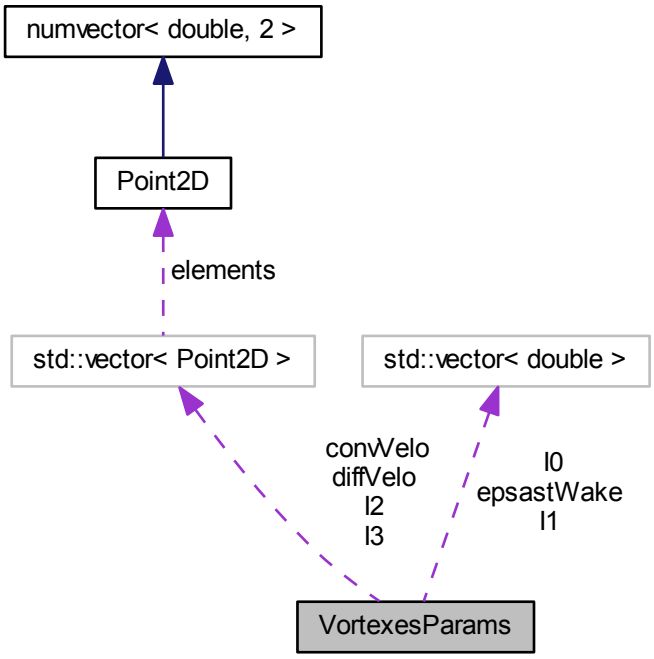
- Vortex2D/Vortex2D.h
- Vortex2D/Vortex2D.cpp

10.34 Структура VortexesParams

Структура, определяющая параметры виртуальных вихрей для отдельного профиля

#include <Velocity.h>

Граф связей класса VortexesParams:



Открытые атрибуты

- std::vector< Point2D > convVelo

  Вектор конвективных скоростей вихрей
- std::vector< Point2D > diffVelo

  Вектор диффузионных скоростей вихрей
- std::vector< double > I0

  Вектор знаменателей (I0) диффузионных скоростей вихрей (обусловленных профилем)
- std::vector< double > I1

  Вектор знаменателей (I1) диффузионных скоростей вихрей (обусловленных завихренностью)
- std::vector< Point2D > I2

  Вектор числителей (I2) диффузионных скоростей вихрей (обусловленных завихренностью)
- std::vector< Point2D > I3

  Вектор числителей (I3) диффузионных скоростей вихрей (обусловленных профилем)
- std::vector< double > epsastWake

  Вектор характерных радиусов вихревых доменов (eps∗)

10.34.1 Подробное описание

Структура, определяющая параметры виртуальных вихрей для отдельного профиля

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Velocity.h строка 60

10.34.2 Данные класса

10.34.2.1 convVelo

std::vector<Point2D> VortexesParams::convVelo

Вектор конвективных скоростей вихрей

См. определение в файле Velocity.h строка 63

10.34.2.2 diffVelo

std::vector<Point2D> VortexesParams::diffVelo

Вектор диффузионных скоростей вихрей

См. определение в файле Velocity.h строка 66

10.34.2.3 epsastWake

std::vector<double> VortexesParams::epsastWake

Вектор характерных радиусов вихревых доменов (eps∗)

См. определение в файле Velocity.h строка 81

10.34.2.4 I0

std::vector<double> VortexesParams::I0

Вектор знаменателей (I0) диффузионных скоростей вихрей (обусловленных профилем)

См. определение в файле Velocity.h строка 69

10.34.2.5 I1

std::vector<double> VortexesParams::I1

Вектор знаменателей (I1) диффузионных скоростей вихрей (обусловленных завихренностью)

См. определение в файле Velocity.h строка 72

10.34.2.6 I2

std::vector<Point2D> VortexesParams::I2

Вектор числителей (I2) диффузионных скоростей вихрей (обусловленных завихренностью)

См. определение в файле Velocity.h строка 75

10.34.2.7 I3

std::vector<Point2D> VortexesParams::I3

Вектор числителей (I3) диффузионных скоростей вихрей (обусловленных профилем)

См. определение в файле Velocity.h строка 78

Объявления и описания членов структуры находятся в файле:

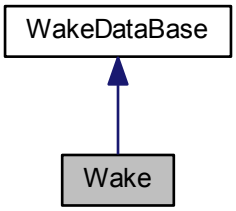
- Velocity/Velocity.h

10.35 Класс Wake

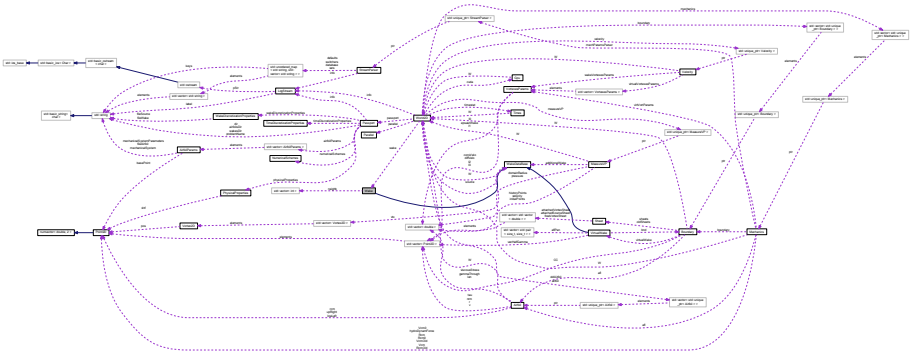
Класс, опеделяющий вихревой след (пелену)

#include <Wake.h>

Граф наследования:Wake:



Граф связей класса Wake:



Открытые члены

- Wake (const World2D &W_)

  Конструктор инициализации
- ~Wake ()

  Деструктор
- void SaveKadr (const std::vector< Vortex2D > &outVtx, const std::string &dir, size_t step, timePeriod &time) const

  Сохранение вихревого следа в файл
- void SaveKadrVtk (const std::vector< Vortex2D > &outVtx, const std::string &dir, size_t step, timePeriod &time) const

  Сохранение вихревого следа в файл .vtk.
- void Inside (const std::vector< Point2D > &newPos, Airfoil &afl, bool isMoves, const Airfoil &oldAfl)

  Проверка пересечения вихрями следа профиля при перемещении

- void Restruct (timePeriod &time)

  Реструктуризация вихревого следа
- int RemoveFar ()

  Зануление далеко улетевших вихрей
- size_t RemoveZero ()

  Исключение нулевых и мелких вихрей
- bool MoveInside (const Point2D &newPos, const Point2D &oldPos, const Airfoil &afl, size_↩ t &panThrough)

  Проверка проникновения точки через границу профиля
- bool MoveInsideMovingBoundary (const Point2D &newPos, const Point2D &oldPos, const Airfoil &oldAfl, const Airfoil &afl, size_t &panThrough)

  Проверка проникновения точки через границу профиля
- void GetPairs (int type)

  Поиск ближайшего соседа
- int Collaps (int type, int times)

  Коллапс вихрей
- void ReadFromFile (const std::string &dir, const std::string &fileName)

  Считывание вихревого следа из файла
- void WakeSynchronize ()

  MPI-синхронизация вихревого следа

Открытые атрибуты

- double collapseRightBorderParameter

  абсцисса, правее которой происходит линейный (вправо) рост радиуса коллапса
- double collapseScaleParameter

  характерный масштаб, на котором происходит рост радиуса коллапса
- const World2D & W

  Константная ссылка на решаемую задачу
- std::vector< Vortex2D > vtx

  Список вихревых элементов

Закрытые данные

- std::vector< int > neighb

  Вектор потенциальных соседей для будущего коллапса

10.35.1 Подробное описание

Класс, опеделяющий вихревой след (пелену)

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Wake.h строка 60

10.35.2 Конструктор(ы)

10.35.2.1 Wake()

Wake::Wake (
const World2D & W_) [inline]

Конструктор инициализации

Аргументы

in	W↩ _	константная ссылка на решаемую задачу

См. определение в файле Wake.h строка 71

```
72      : WakeDataBase(W_) { };
```

10.35.2.2 ∼Wake()

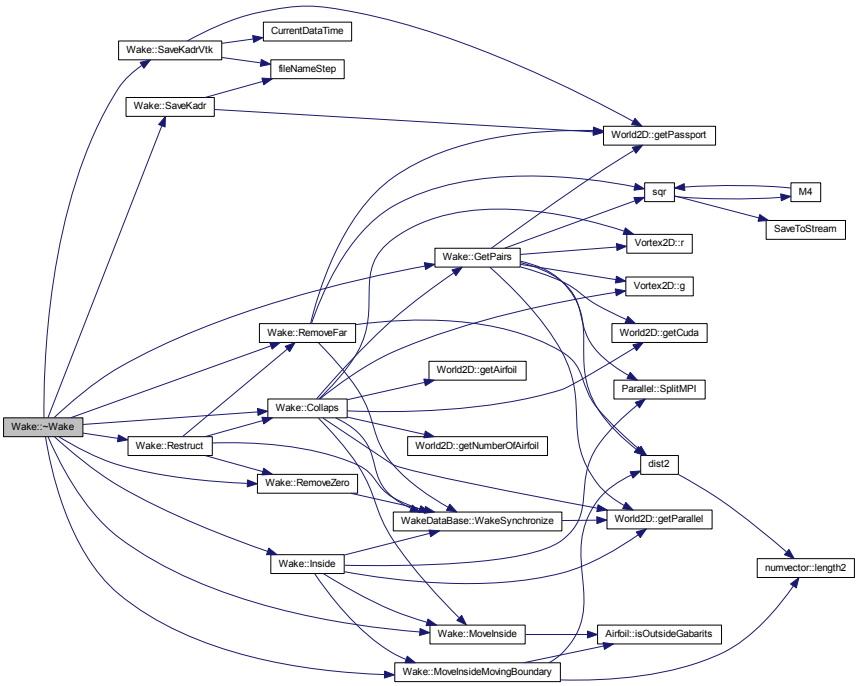
Wake::∼Wake ( ) [inline]

Деструктор

См. определение в файле Wake.h строка 75

```
75 { };
```

Граф вызовов:



10.35.3 Методы

10.35.3.1 Collaps()

int Wake::Collaps (
int type,
int times)

Коллапс вихрей

Аргументы

in	type	тип коллапса: • 0 — без приоритета знаков • 1 — коллапсировать только вихри разных знаков • 2 — коллапсировать только вихри одного знака
in	times	число проходов алгоритма коллапса

Возвращает

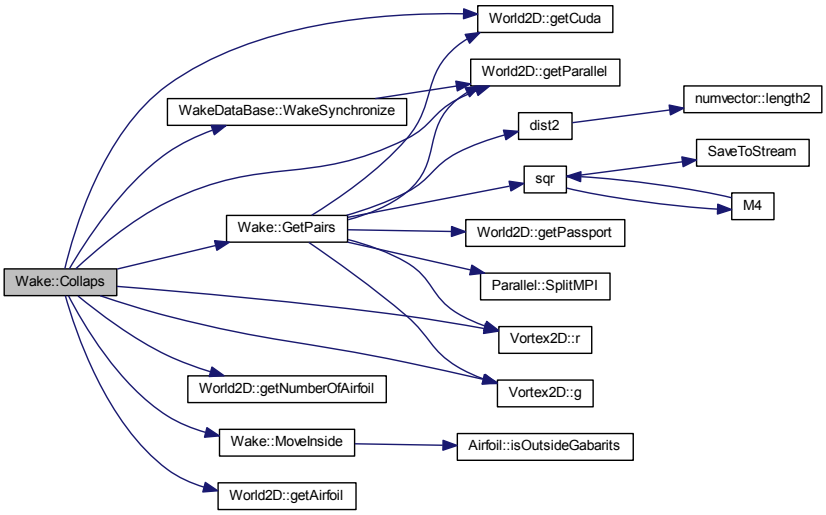
число зануленных вихрей

См. определение в файле Wake.cpp строка 462

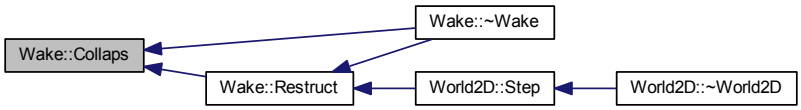
```
463 {
464     int nHlop = 0; //общее число убитых вихрей
465
466     //int loc_hlop = 0; //
467
468     std::vector<bool> flag; //как только вихрь сколлапсировался  flag = 1
469     for (int z = 0; z < times; ++z)
470     {
471
472 #if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_PAIRS))
473         //W.getInfo('t') << "nvtx (" << parallel.myidWork << ") = " << vtx.size() << std::endl;
474         const_cast<Gpu&>(W.getCuda()).RefreshWake();
475         GPUGetPairs(type);
476 #else
477         GetPairs(type);
478 #endif
479
480         if (W.getParallel().myidWork == 0)
481         {
482             //loc_hlop = 0;//число схлопнутых вихрей
483
484             flag.clear();
485             flag.resize(vtx.size(), false);
486
487             double sumAbsGam, iws;
488             Point2D newPos;
489
490             for (size_t vt = 0; vt < vtx.size() - 1; ++vt)
491             {
492                 Vortex2D& vtxI = vtx[vt];
493
494                 if (!flag[vt])
495                 {
496                     int ssd = neighb[vt];
497                     if ((ssd != 0) && (!flag[ssd]))
498                     {
499                         Vortex2D& vtxK = vtx[ssd];
500
501                         flag[ssd] = true;
502
503                         Vortex2D sumVtx;
504                         sumVtx.g() = vtxI.g() + vtxK.g();
505
506                         switch (type)
507                         {
508                         case 0:
509                         case 2:
510                             sumAbsGam = fabs(vtxI.g()) + fabs(vtxK.g());
511
512                             iws = sumAbsGam > 1e-10 ? 1.0 / sumAbsGam : 1.0;
513
514                             sumVtx.r() = (vtxI.r()*fabs(vtxI.g()) + vtxK.r()*fabs(vtxK.
    g())) * iws;
515                             break;
516
517                         case 1:
518                             sumVtx.r() = (fabs(vtxI.g()) > fabs(vtxK.g())) ? vtxI.
    r() : vtxK.r();
519                             break;
520                         }
521
522                         bool fl_hit = true;
523                         //double Ch1[2];
524                         size_t hitpan = -1;
525
526                         for(size_t afl = 0; afl < W.getNumberOfAirfoil(); afl++)
527                         {
528                             //проверим, не оказался ли новый вихрь внутри контура
529                             if (MoveInside(sumVtx.r(), vtxI.r(), W.
    getAirfoil(afl), hitpan) || MoveInside(sumVtx.r(), vtxK.r(),
    W.getAirfoil(afl), hitpan))
530                                 fl_hit = false;
531                         }//for
532
533                         if (fl_hit)
534                         {
535                             vtxI = sumVtx;
536                             vtxK.g() = 0.0;
537                             nHlop++;
538                         }//if (fl_hit)
539
540                     }//if ((ssd!=0)&&(!flag[ssd]))
541                 }//if !flag[vt]
542             }//for vt
```

```
        }

        WakeSynchronize();
    }//for z

    return nHlop;
}//Collaps(...)
```

Граф вызовов:



Граф вызова функции:



10.35.3.2 GetPairs()

void Wake::GetPairs (
int type)

Поиск ближайшего соседа

Аргументы

in	type	тип коллапса: • 0 — без приоритета знаков • 1 — коллапсировать только вихри разных знаков • 2 — коллапсировать только вихри одного знака

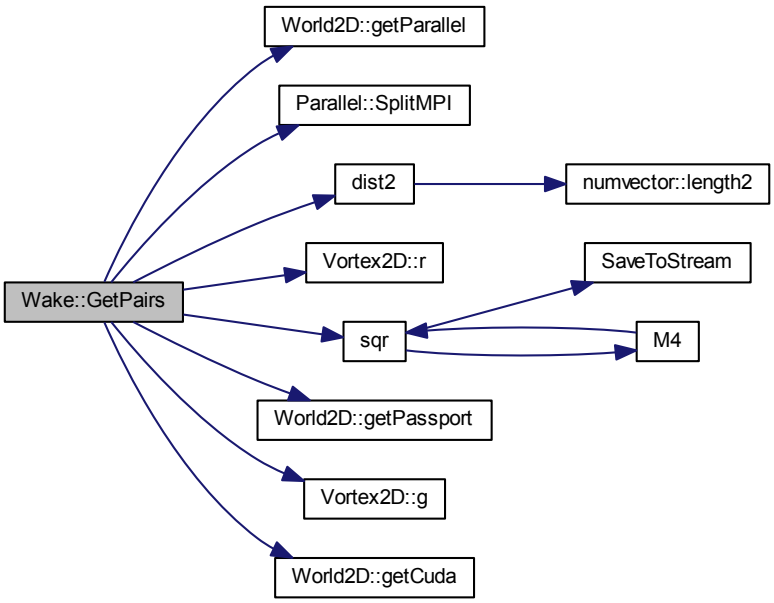
Необходимо сделать Временно для профиля из 1000 панелей

См. определение в файле Wake.cpp строка 349

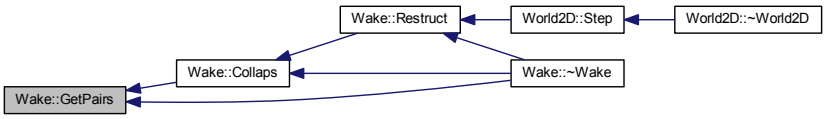
```
350 {
351     int id = W.getParallel().myidWork;
352     parProp par = W.getParallel().SplitMPI(vtx.size());
353
355     const double max_g = 0.03;      //максимальная циркуляция вихря, получаемого на первом шаге расчета
356     const double coeff_max_g = 0.25; // коэффициент, определяющий максимально возможную циркуляцию вихря
    при коллапсе
357
358     std::vector<int> locNeighb; //локальный массив соседей (для данного процессора)
359     locNeighb.resize(par.myLen);
360
361     Point2D Ri, Rk;
362
363 #pragma omp parallel for default(none) shared(type, locNeighb, par) schedule(dynamic, DYN_SCHEDULE)
364     for (int locI = 0; locI < par.myLen; ++locI)
365     {
366         int s = locI + par.myDisp;
367         const Vortex2D& vtxI = vtx[s];
368
369         locNeighb[locI] = 0;//по умолчанию
370
371         bool found = false;
372
373         double r2, r2test;
374
375         const double& cSP = collapseScaleParameter;
376         const double& cRBP = collapseRightBorderParameter;
377
378         while ( (!found) && ( s < vtx.size() -1 ) )
379         {
380             s++;
381             const Vortex2D& vtxK = vtx[s];
382
383             r2 = dist2(vtxI.r(), vtxK.r());
384
385             //линейное увеличение радиуса коллапса
386             double mnog = std::max(1.0, 2.0 * (vtxI.r()[0] - cRBP) / cSP);
387
388             r2test = sqr( W.getPassport().
    wakeDiscretizationProperties.epscol * mnog );
389
390             if (type == 1)
391                 r2test *= 4.0; //Увеличение радиуса коллапса в 2 раза для коллапса вихрей разных знаков
392
393             if (r2<r2test)
394             {
395                 switch (type)
396                 {
397                     case 0:
398                         found = ( fabs(vtxI.g()*vtxK.g()) != 0.0) && (fabs(vtxI.
    g() + vtxK.g()) < max_g*coeff_max_g * mnog);
399                         break;
400                     case 1:
401                         found = (vtxI.g()*vtxK.g() < 0.0);
402                         break;
403                     case 2:
404                         found = (vtxI.g()*vtxK.g() > 0.0) && (fabs(vtxI.g() + vtxK.
    g()) < max_g*coeff_max_g * mnog);
405                         break;
406                 }
407             }//if r2<r2_test
408         }//while
```

```
        if (found)
            locNeighb[locI] = s;
    }//for locI

    if (id == 0)
        neighb.resize(vtx.size());

    MPI_Gatherv(locNeighb.data(), par.myLen, MPI_INT, neighb.data(), par.len.data(), par.disp.data(), MPI_INT, 0, W.getParallel().commWork);

}//GetPairs(...)
```

Граф вызовов:



Граф вызова функции:



10.35.3.3 MoveInside()

bool Wake::MoveInside (
    const Point2D & newPos,

```
const Point2D & oldPos,
const Airfoil & afl,
size_t & panThrough )
```

Проверка проникновения точки через границу профиля

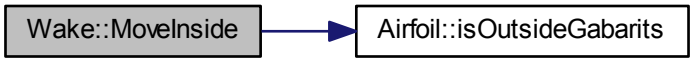
Аргументы

in	newPos	константная ссылка на смещенное (новое) положение
in	oldPos	константная ссылка на несмещенное (старое) положение
in	afl	константная ссылка на контролируемый профиль
out	panThrough	номер "протыкаемой" панели return признак пересечения профиля

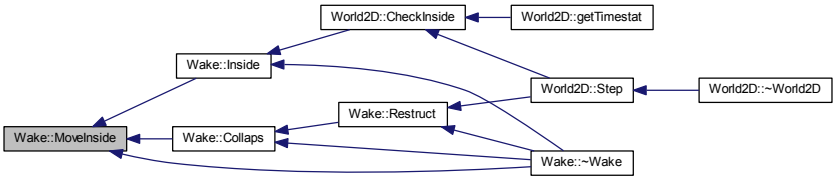
См. определение в файле Wake.cpp строка 172

```
173 {
174
175     const double porog_r = 1e-12;
176
177     double minDist = 1.0E+10; //расстояние до пробиваемой панели
178     panThrough = -1;
179
180     //проверка габ. прямоугольника
181     if (afl.isOutsideGabarits(newPos) && afl.isOutsideGabarits(oldPos))
182         return false;
183
184     //если внутри габ. прямоугольника - проводим контроль
185     bool hit = false;
186
187     //Определение прямой: Ax+By+D=0 - перемещение вихря
188     double A = newPos[1] - oldPos[1];
189     double B = oldPos[0] - newPos[0];
190     double D = oldPos[1] * newPos[0] - oldPos[0] * newPos[1];
191
192     double A1, B1, D1;
193
194     //Проверка на пересечение
195     double r0 = 0, r1 = 0, r2 = 0, r3 = 0;
196     for (size_t j = 0; j < afl.np; ++j)
197     {
198         r0 = A*afl.r[j][0] + B*afl.r[j][1] + D;
199         r1 = A*afl.r[j + 1][0] + B*afl.r[j + 1][1] + D;
200
201         if (fabs(r0) < porog_r) r0 = 0.0;
202         if (fabs(r1)<porog_r) r1 = 0.0;
203
204         if (r0*r1>0)
205             continue;
206
207         //Определение прямой:A1x+B1y+D1=0 - панель
208         A1 = afl.r[j + 1][1] - afl.r[j][1];
209         B1 = afl.r[j][0] - afl.r[j + 1][0];
210         D1 = afl.r[j][1] * afl.r[j + 1][0] - afl.r[j][0] * afl.r[j + 1][1];
211
212         r2 = A1*oldPos[0] + B1*oldPos[1] + D1;
213         r3 = A1*newPos[0] + B1*newPos[1] + D1;
214
215         if (fabs(r2) < porog_r) r2 = 0.0;
216         if (fabs(r3) < porog_r) r3 = 0.0;
217
218         if (r2*r3 > 0)
219             continue;
220
221         hit = true;// пробила!
222         double d2 = (oldPos[0] - (B*D1 - D*B1) / (A*B1 - B*A1))*(oldPos[0] - (B*D1 - D*B1) / (A*B1 - B*A1))
    + \
223             (oldPos[1] - (A1*D - D1*A) / (A*B1 - B*A1))*(oldPos[1] - (A1*D - D1*A) / (A*B1 - B*A1));
224
225         if (d2 < minDist)
226         {
227             minDist = d2;
228             panThrough = j;
229         }//if d2
230     }//for j
231
232     return hit;
233 }//MoveInside(...)
```

Граф вызовов:



Граф вызова функции:



10.35.3.4 MoveInsideMovingBoundary()

```
bool Wake::MoveInsideMovingBoundary (
            const Point2D & newPos,
            const Point2D & oldPos,
            const Airfoil & oldAfl,
            const Airfoil & afl,
            size_t & panThrough )
```

Проверка проникновения точки через границу профиля

Аргументы

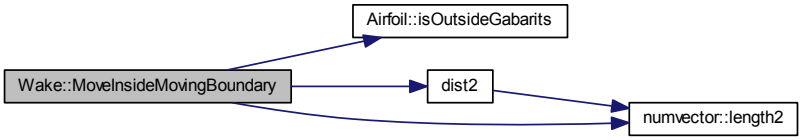
in	newPos	константная ссылка на смещенное (новое) положение вихря
in	oldPos	константная ссылка на несмещенное (старое) положение вихря
in	oldAfl	константная ссылка на состояние контролируемого профиля до перемещения
in	afl	константная ссылка на контролируемый профиль
out	panThrough	номер "протыкаемой" панели return признак пересечения профиля

Необходимо сделать сравнить производительности двух inside-ов

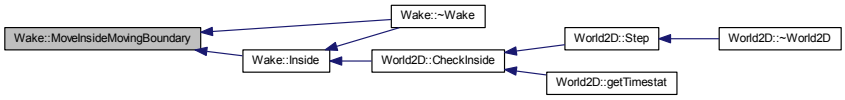
См. определение в файле Wake.cpp строка 235

```
{
    panThrough = -1;


    //проверка габ. прямоугольника
    if (afl.isOutsideGabarits(newPos) && afl.isOutsideGabarits(oldPos))
        return false;

    bool hit = false;

    double angle = 0;
    double cs, sn;

    double dist2 = 1000000000.0;

    for (int i = 0; i<afl.np; i++)
    {
        Point2D v1, v2, vv;
        v1 = afl.r[i] - newPos;

        v2 = afl.r[i + 1] - newPos;

        vv = afl.r[i + 1] - afl.r[i];

        double dst = v1.length2() + v2.length2();
        if (dst < dist2)
        {
            dist2 = dst;
            panThrough = i;
        }

        cs = v1 * v2;
        sn = v1 ^ v2;

        angle += atan2(sn, cs);
    }//for i

    hit = ((angle > 3.00) || (angle < -3.00));

    return hit;
}//MoveInsideMovingBoundary(...)
```

Граф вызовов:



Граф вызова функции:



10.35.3.5 ReadFromFile()

```
void WakeDataBase::ReadFromFile (
            const std::string & dir,
            const std::string & fileName )   [inherited]
```

Считывание вихревого следа из файла

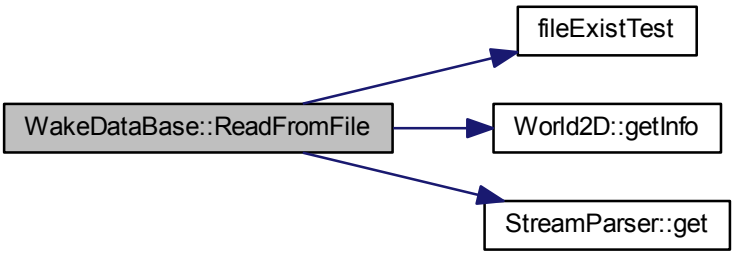
Аргументы

in	dir	константная ссылка на строку, задающую каталог, где лежит файл с вихревым следом
in	fileName	константная ссылка на строку, задающую имя файла с вихревым следом

См. определение в файле WakeDataBase.cpp строка 53

```
54 {
55     std::string filename = dir + fileName;
56     std::ifstream wakeFile;
57
58     if (fileExistTest(filename, W.getInfo()))
59     {
60         std::stringstream wakeFile(Preprocessor(filename).resultString);
61
62         LogStream XXX;
63         StreamParser wakeParser(XXX, "vortex wake file parser", wakeFile);
64
65         wakeParser.get("vtx", vtx);
66     }
67 }//ReadFromFile(...)
```

Граф вызовов:



Граф вызова функции:

10.35.3.6 RemoveFar()

int Wake::RemoveFar ( )

Зануление далеко улетевших вихрей

Возвращает

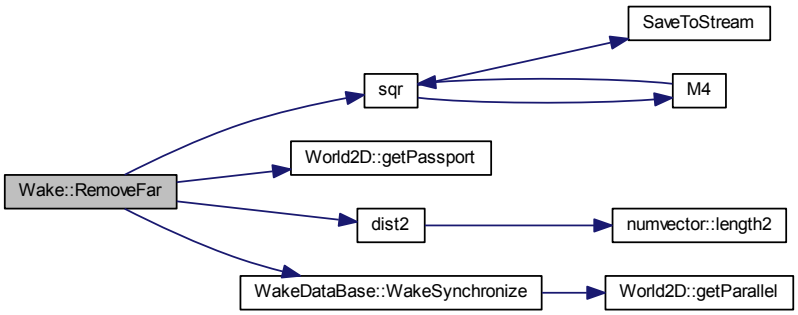
число вихрей в дальнем следе, которые занулены

Необходимо сделать Пока профиль 1, расстояние от его центра; сделать от самого правого профиля

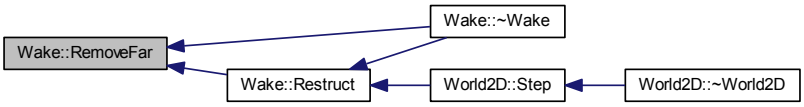
См. определение в файле Wake.cpp строка 552

```
553 {
554     int nFar = 0;
555     double distFar2 = sqr(W.getPassport().
    wakeDiscretizationProperties.distFar);
557     Point2D zerovec = { 0.0, 0.0 };
558 #pragma omp parallel for default(none) shared(distFar2, zerovec) reduction(+:nFar)
559     for (int i = 0; i <static_cast<int>(vtx.size()); ++i)
560     {
561         if (dist2(vtx[i].r(), zerovec) > distFar2)
562         {
563             vtx[i].g() = 0.0;
564             nFar++;
565         }
566     }
567
568     WakeSynchronize();
569     return nFar;
570 }//RemoveFar()
```

Граф вызовов:



Граф вызова функции:



10.35.3.7 RemoveZero()

size_t Wake::RemoveZero ( )

Исключение нулевых и мелких вихрей

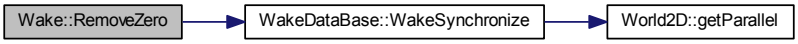
Возвращает

число исключенных вихрей

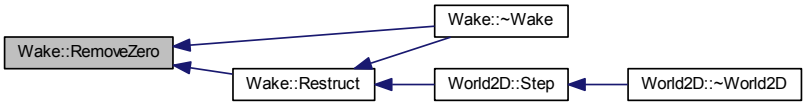
См. определение в файле Wake.cpp строка 573

```
574 {
575     const double porog_g = 1e-15;
576
577     std::vector<Vortex2D> newWake;
578
579     newWake.reserve(vtx.size());
580
581     for (size_t q = 0; q < vtx.size(); ++q)
582     if (fabs(vtx[q].g()) > porog_g)
583         newWake.push_back(vtx[q]);
584
585     size_t delta = vtx.size() - newWake.size();
586
587     newWake.swap(vtx);
588
589     WakeSynchronize();
590
591     return delta;
592 }//RemoveZero()
```

Граф вызовов:



Граф вызова функции:



10.35.3.8 Restruct()

void Wake::Restruct (
timePeriod & time)

Реструктуризация вихревого следа

Исполняется сразу для всех вихрей в пелене
Вихри, находящиеся далеко от профилей, удаляются
Вихри, которые сильно сблизились, коллапсируются

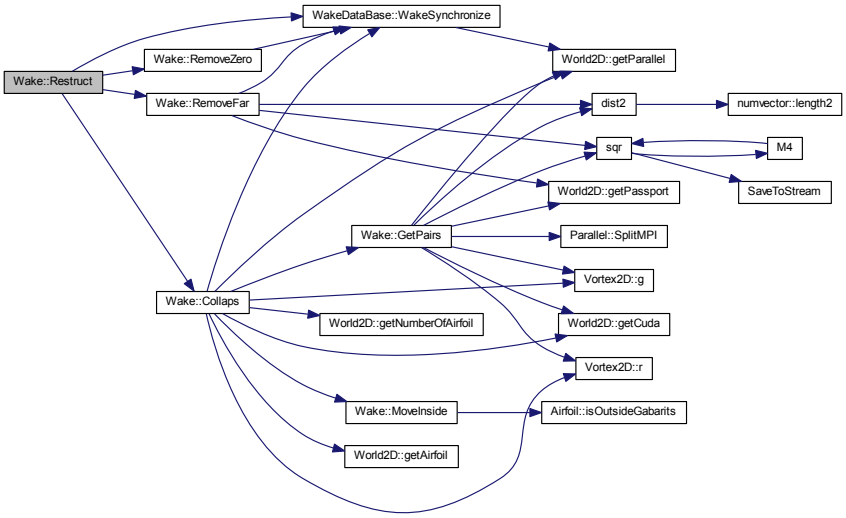
Аргументы

out	time	ссылка на промежуток времени — пару чисел (время начала и время конца операции)

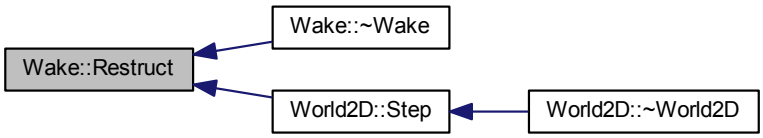
См. определение в файле Wake.cpp строка 596

```
597 {
598     time.first = omp_get_wtime();
599
600     WakeSynchronize();
601
602     Collaps(1, 1);
603     Collaps(2, 1);
604
605     RemoveFar();
606     RemoveZero();
607
608     time.second = omp_get_wtime();
609 }//Restruct(...)
```

Граф вызовов:



Граф вызова функции:



10.35.3.9 SaveKadr()

```
void Wake::SaveKadr (
            const std::vector< Vortex2D > & outVtx,
            const std::string & dir,
            size_t step,
            timePeriod & time ) const
```

Сохранение вихревого следа в файл

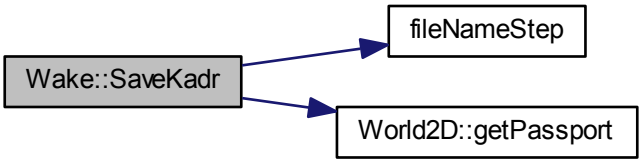
Аргументы

in	outVtx	константная ссылка на вектор вихрей, которые нужно вывести в файл
in	dir	константная ссылка на строку, задающую каталог, куда сохранять файл с вихревым следом
in	step	номер кадра для сохранения
out	time	ссылка на промежуток времени — пару чисел (время начала и время конца операции)

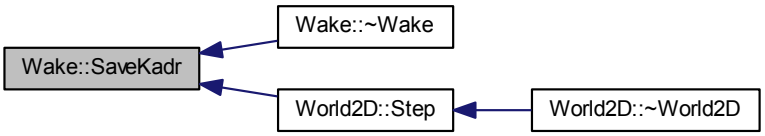
См. определение в файле Wake.cpp строка 54

```
55 {
56     time.first = omp_get_wtime();
57
58     std::string fname = fileNameStep("Kadr", W.getPassport().
   timeDiscretizationProperties.nameLength, step, "txt");
59
60     std::ofstream outfile;
61     size_t numberNonZero = 0;
62
63     for (size_t q = 0; q < outVtx.size(); ++q)
64     {
65         if (outVtx[q].g() != 0.0)
66             numberNonZero++;
67     }
68
69 #if defined(_WIN32)
70     _mkdir((dir + "snapshots").c_str());
71 #else
72     mkdir((dir + "snapshots").c_str(), S_IRWXU | S_IRGRP | S_IROTH);
73 #endif
74
75     outfile.open(dir + "snapshots/" + fname);
76 //  PrintLogoToTextFile(outfile, dir + "snapshots/" + fname, "Positions and circulations of vortices in the
   wake");
77
78 //  PrintHeaderToTextFile(outfile, "Number of vortices");
79     outfile << numberNonZero << std::endl; //Сохранение числа вихрей в пелене
80 //  outfile << std::endl << "// " << numberNonZero << std::endl; //Сохранение числа вихрей в пелене
81
82 //  PrintHeaderToTextFile(outfile, "x_i     y_i     G_i");
83
84     for (size_t i = 0; i < outVtx.size(); i++)
85     {
86         const Point2D& r = outVtx[i].r();
87         double gi = outVtx[i].g();
88
89         if (gi != 0.0)
90         {
91             outfile << static_cast<int>(i) << " " << W.getPassport().
   wakeDiscretizationProperties.eps << " " << r[0] << " " << r[1] << " " << "
   0.0" << " " << "0.0" << " " << "0.0" << " " << gi << std::endl;
92             //нули пишутся для совместимости с трехмерной программой и обработчиком ConMDV
93             //outfile << std::endl << xi << " " << yi << " " << gi;
94         }
95     }//for i
96     outfile.close();
97
98     time.second = omp_get_wtime();
99 }//SaveKadr(...)
```

Граф вызовов:



Граф вызова функции:



10.35.3.10 SaveKadrVtk()

```
void Wake::SaveKadrVtk (
            const std::vector< Vortex2D > & outVtx,
            const std::string & dir,
            size_t step,
            timePeriod & time ) const
```

Сохранение вихревого следа в файл .vtk.

Аргументы

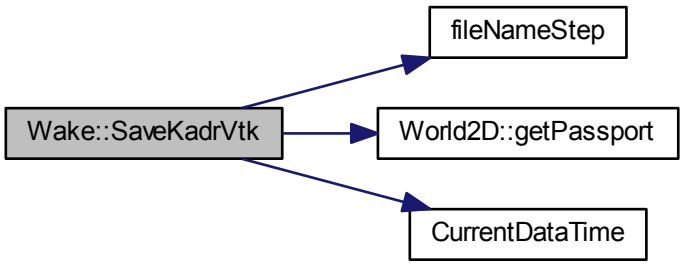
in	outVtx	константная ссылка на вектор вихрей, которые нужно вывести в файл
in	dir	константная ссылка на строку, задающую каталог, куда сохранять файл с вихревым следом
in	step	номер кадра для сохранения
out	time	ссылка на промежуток времени — пару чисел (время начала и время конца операции)

См. определение в файле Wake.cpp строка 103

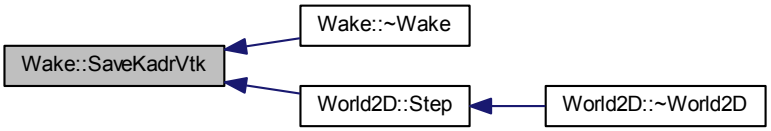
```
104 {
```

```
    time.first = omp_get_wtime();

    std::string fname = fileNameStep("Kadr", W.getPassport().timeDiscretizationProperties.nameLength, step, "vtk");

    std::ofstream outfile;
    size_t numberNonZero = 0;

    for (size_t q = 0; q < outVtx.size(); ++q)
    {
        //if (outVtx[q].g() != 0.0)
            numberNonZero++;
    }

#if defined(_WIN32)
    _mkdir((dir + "snapshots").c_str());
#else
    mkdir((dir + "snapshots").c_str(), S_IRWXU | S_IRGRP | S_IROTH);
#endif

    outfile.open(dir + "snapshots/" + fname);

    outfile << "# vtk DataFile Version 2.0" << std::endl;
    outfile << "VM2D VTK result: " << (dir + "snapshots/" + fname).c_str() << " saved " << CurrentDataTime() << std::endl;
    outfile << "ASCII" << std::endl;
    outfile << "DATASET UNSTRUCTURED_GRID" << std::endl;
    outfile << "POINTS " << numberNonZero << " float" << std::endl;


    for (size_t i = 0; i < outVtx.size(); i++)
    {
        Point2D rrr = outVtx[i].r();

        double xi = (outVtx[i].r())[0];
        double yi = (outVtx[i].r())[1];
        double gi =outVtx[i].g();

        //if (gi != 0.0)
        {
            outfile << xi << " " << yi << " " << "0.0"  << std::endl;
        }
    }//for i

    outfile << "CELLS " << numberNonZero << " " << 2*numberNonZero << std::endl;
    for (size_t i=0; i<numberNonZero; ++i)
        outfile << "1 " << i << std::endl;

    outfile << "CELL_TYPES " << numberNonZero << std::endl;
    for (size_t i=0; i<numberNonZero; ++i)
        outfile << "1" << std::endl;

    outfile << std::endl;
    outfile << "POINT_DATA " << numberNonZero << std::endl;
    outfile << "SCALARS Gamma float 1" << std::endl;
    outfile << "LOOKUP_TABLE default" << std::endl;

    for (size_t i = 0; i < outVtx.size(); i++)
    {
        outfile << outVtx[i].g() << std::endl;
    }//for i


    outfile.close();

    time.second = omp_get_wtime();
}//SaveKadrVtk(...)
```

Граф вызовов:



Граф вызова функции:



10.35.4 Данные класса

10.35.4.1 collapseRightBorderParameter

double Wake::collapseRightBorderParameter

абсцисса, правее которой происходит линейный (вправо) рост радиуса коллапса

См. определение в файле Wake.h строка 163

10.35.4.2 collapseScaleParameter

double Wake::collapseScaleParameter

характерный масштаб, на котором происходит рост радиуса коллапса

См. определение в файле Wake.h строка 166

10.35.4.3 neighb

std::vector<int> Wake::neighb [private]

Вектор потенциальных соседей для будущего коллапса

См. определение в файле Wake.h строка 64

10.35.4.4 vtx

std::vector<Vortex2D> WakeDataBase::vtx [inherited]

Список вихревых элементов

См. определение в файле WakeDataBase.h строка 61

10.35.4.5 W

const World2D& WakeDataBase::W [inherited]

Константная ссылка на решаемую задачу

См. определение в файле WakeDataBase.h строка 55

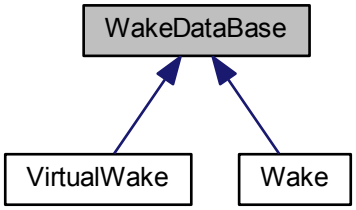
Объявления и описания членов классов находятся в файлах:

- Wake/Wake.h
- Wake/Wake.cpp

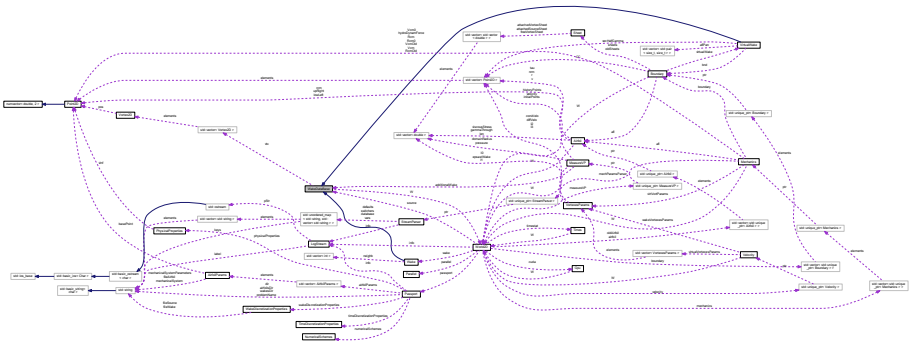
10.36 Класс WakeDataBase

#include <WakeDataBase.h>

Граф наследования:WakeDataBase:



Граф связей класса WakeDataBase:



Открытые члены

- WakeDataBase (const World2D &W_)

  Конструктор инициализации
- virtual ~WakeDataBase ()
- void ReadFromFile (const std::string &dir, const std::string &fileName)

  Считывание вихревого следа из файла
- void WakeSynchronize ()

  MPI-синхронизация вихревого следа

Открытые атрибуты

- const World2D & W

  Константная ссылка на решаемую задачу
- std::vector< Vortex2D > vtx

  Список вихревых элементов

10.36.1 Подробное описание

См. определение в файле WakeDataBase.h строка 50

10.36.2 Конструктор(ы)

10.36.2.1 WakeDataBase()

WakeDataBase::WakeDataBase (
const World2D & W_) [inline]

Конструктор инициализации

Аргументы

in	W↩ _	константная ссылка на решаемую задачу

См. определение в файле WakeDataBase.h строка 60

```
61        : W(W_) { };
```

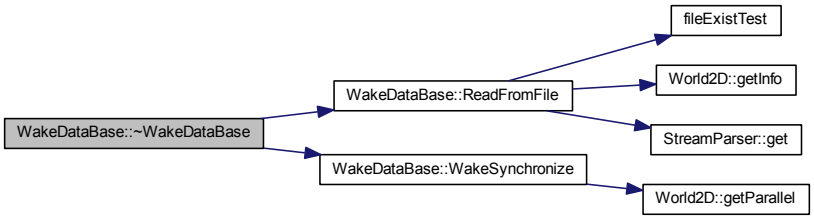
10.36.2.2 ~WakeDataBase()

virtual WakeDataBase::~WakeDataBase ( ) [inline], [virtual]

См. определение в файле WakeDataBase.h строка 101

```
101 {};
```

Граф вызовов:



10.36.3 Методы

10.36.3.1 ReadFromFile()

void WakeDataBase::ReadFromFile (
const std::string & dir,
const std::string & fileName)

Считывание вихревого следа из файла

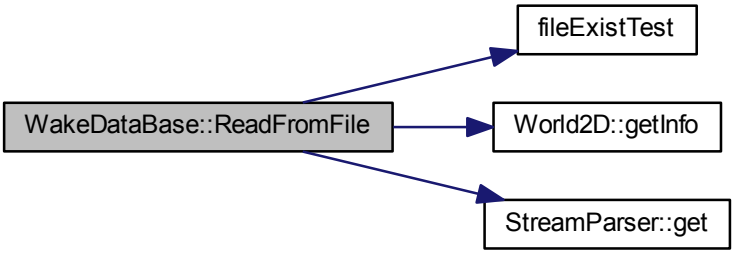
Аргументы

in	dir	константная ссылка на строку, задающую каталог, где лежит файл с вихревым следом
in	fileName	константная ссылка на строку, задающую имя файла с вихревым следом

См. определение в файле WakeDataBase.cpp строка 53

```
54 {
55     std::string filename = dir + fileName;
56     std::ifstream wakeFile;
57
58     if (fileExistTest(filename, W.getInfo()))
59     {
60         std::stringstream wakeFile(Preprocessor(filename).resultString);
61
62         LogStream XXX;
63         StreamParser wakeParser(XXX, "vortex wake file parser", wakeFile);
64
65         wakeParser.get("vtx", vtx);
66     }
67 }//ReadFromFile(...)
```

Граф вызовов:



Граф вызова функции:

10.36.4 Данные класса

10.36.4.1 vtx

std::vector<Vortex2D> WakeDataBase::vtx

Список вихревых элементов

См. определение в файле WakeDataBase.h строка 61

10.36.4.2 W

const World2D& WakeDataBase::W

Константная ссылка на решаемую задачу

См. определение в файле WakeDataBase.h строка 55

Объявления и описания членов классов находятся в файлах:

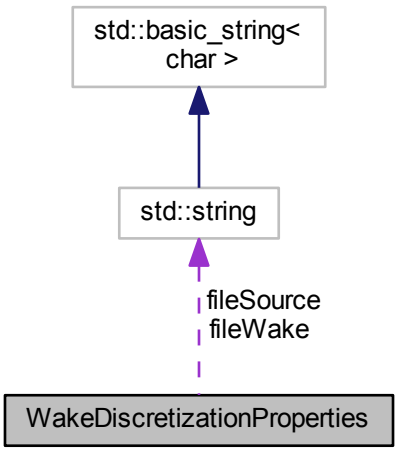
- WakeDataBase/WakeDataBase.h
- WakeDataBase/WakeDataBase.cpp

10.37 Структура WakeDiscretizationProperties

Структура, задающая параметры параметры дискретизации вихревого следа

#include <Passport.h>

Граф связей класса WakeDiscretizationProperties:



Открытые атрибуты

- double eps

  Радиус вихря
- double eps2

  Квадрат радиуса вихря
- double epscol

  Радиус коллапса
- double distFar

  Расстояние от центра самого подветренного (правого) профиля, на котором вихри уничтожаются
- double delta

  Расстояние, на которое рождаемый вихрь отодвигается от профиля
- int vortexPerPanel

  Число вихрей, рождаемых на каждой панели профииля
- std::string fileWake

  Имя файла с начальным состоянием вихревого следа (без полного пути)
- std::string fileSource

  Имя файла с положениями источников (без полного пути)

10.37.1 Подробное описание

Структура, задающая параметры параметры дискретизации вихревого следа

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле Passport.h строка 171

10.37.2 Данные класса

10.37.2.1 delta

double WakeDiscretizationProperties::delta

Расстояние, на которое рождаемый вихрь отодвигается от профиля

См. определение в файле Passport.h строка 186

10.37.2.2 distFar

double WakeDiscretizationProperties::distFar

Расстояние от центра самого подветренного (правого) профиля, на котором вихри уничтожаются

См. определение в файле Passport.h строка 183

10.37.2.3 eps

double WakeDiscretizationProperties::eps

Радиус вихря

См. определение в файле Passport.h строка 174

10.37.2.4 eps2

double WakeDiscretizationProperties::eps2

Квадрат радиуса вихря

См. определение в файле Passport.h строка 177

10.37.2.5 epscol

double WakeDiscretizationProperties::epscol

Радиус коллапса

См. определение в файле Passport.h строка 180

10.37.2.6 fileSource

std::string WakeDiscretizationProperties::fileSource

Имя файла с положениями источников (без полного пути)

См. определение в файле Passport.h строка 195

10.37.2.7 fileWake

std::string WakeDiscretizationProperties::fileWake

Имя файла с начальным состоянием вихревого следа (без полного пути)

См. определение в файле Passport.h строка 192

10.37.2.8 vortexPerPanel

int WakeDiscretizationProperties::vortexPerPanel

Число вихрей, рождаемых на каждой панели профииля

См. определение в файле Passport.h строка 189

Объявления и описания членов структуры находятся в файле:

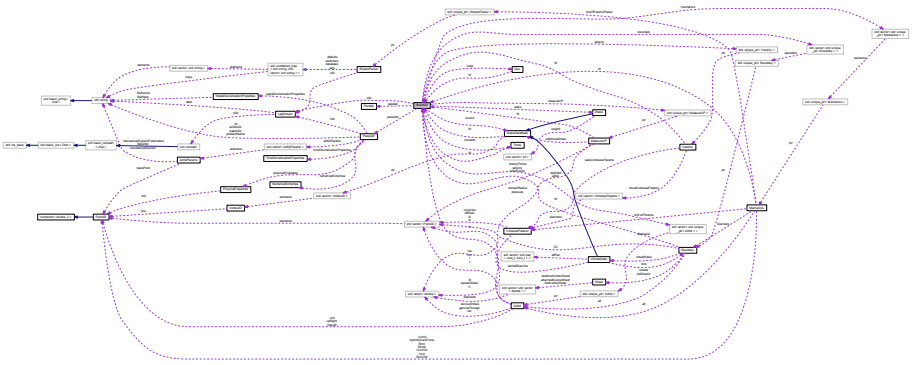
- Passport/Passport.h

10.38 Класс World2D

Класс, опеделяющий текущую решаемую задачу

#include <World2D.h>

Граф связей класса World2D:



Открытые члены

- LogStream & getInfo () const

  Возврат ссылки на объект LogStream Используется в техничеких целях для организации вывода
- std::ostream & getInfo (char x) const

  Возврат ссылки на поток вывода информации Необходимо для вывода телеметрической информации, информации об ошибках и т.п.
- const Airfoil & getAirfoil (size_t i) const

  Возврат константной ссылки на объект профиля
- Airfoil & getNonConstAirfoil (size_t i) const

  Возврат неконстантной ссылки на объект профиля
- size_t getNumberOfAirfoil () const

  Возврат количества профилей в задаче
- const Boundary & getBoundary (size_t i) const

  Возврат константной ссылки на объект граничного условия
- Boundary & getNonConstBoundary (size_t i) const

  Возврат неконстантной ссылки на объект граничного условия
- size_t getNumberOfBoundary () const

  Возврат количества граничных условий в задаче
- const Wake & getWake () const

  Возврат константной ссылки на вихревой след
- const WakeDataBase & getSource () const

  Возврат константной ссылки на источники в области течения
- const Velocity & getVelocity () const

  Возврат константной ссылки на объект для вычисления скоростей
- Velocity & getNonConstVelocity () const

  Возврат неконстантной ссылки на объект для вычисления скоростей
- const Passport & getPassport () const

  Возврат константной ссылки на паспорт
- const Parallel & getParallel () const

  Возврат константной ссылки на параметры распараллеливания по MPI.
- const Gpu & getCuda () const

  Возврат константной ссылки на объект, связанный с видеокартой (GPU)
- size_t getCurrentStep () const

  Возврат номера текущего временного шага
- const Times & getTimestat () const

  Возврат константной ссылки на временную статистику выполнения шага расчета по времени
- void SolveLinearSystem (timePeriod &time)

  Решение системы линейных алгебраических уравнений
- void FillMatrixAndRhs (timePeriod &time)

  Заполнение матрицы системы для всех профилей
- void ReserveMemoryForMatrixAndRhs (timePeriod &time)

  Вычисляем размер матрицы и резервируем память под нее и под правую часть
- void CalcVortexVelo (double dt, timePeriod &convTime, timePeriod &diffTime)

  Вычисляем скорости вихрей (в пелене и виртуальных)
- void MoveVortexes (double dt, std::vector< Point2D > &newPos, timePeriod &time)

  Вычисляем новые положения вихрей (в пелене и виртуальных)
- void CheckInside (std::vector< Point2D > &newPos, timePeriod &time)

  Проверка проникновения вихрей внутрь профиля
- World2D (const Passport &passport_, const Parallel &parallel_)

  Конструктор

- ~World2D ()

  Деструктор
- bool isFinished () const

  Функция, возвращающая признак завершения счета в решаемой задаче
- void Step ()

  Основная функция выполнения одного шага по времени
- void GenerateMechanicsHeader (size_t mechanicsNumber)

Закрытые данные

- LogStream info

  Поток для вывода логов и сообщений об ошибках
- std::vector< std::unique_ptr< Airfoil > > airfoil

  Список умных казателей на обтекаемые профили
- std::vector< std::unique_ptr< Airfoil > > oldAirfoil

  Список умных казателей на обтекаемые профили для сохранения старого положения
- std::vector< std::unique_ptr< Boundary > > boundary

  Список умных указателей на формирователи граничных условий на профилях
- std::vector< std::unique_ptr< Mechanics > > mechanics

  Список умных указателей на типы механической системы для каждого профиля
- std::unique_ptr< Velocity > velocity

  Умный укзатель на объект, определяющий методику вычисления скоростей
- Wake wake

  Вихревой след
- WakeDataBase source

  Источники
- std::unique_ptr< MeasureVP > measureVP

  Поле скоростей и давления (для сохранения в файл)
- Eigen::MatrixXd matr

  Матрица системы
- Eigen::MatrixXd invMatr

  Обратная матрица
- Eigen::VectorXd rhs

  Правая часть системы
- Eigen::VectorXd sol

  Решение системы
- const Passport & passport

  Константная ссылка на паспорт конкретного расчета
- const Parallel & parallel

  Константная ссылка на параметры исполнения задачи в параллельном MPI-режиме
- Gpu cuda

  Объект, управляющий графическим ускорителем
- size_t currentStep

  Текущий номер шага в решаемой задаче
- Times timestat

  Сведения о временах выполнения основных операций

10.38.1 Подробное описание

Класс, опеделяющий текущую решаемую задачу

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

См. определение в файле World2D.h строка 83

10.38.2 Конструктор(ы)

10.38.2.1 World2D()

```
World2D::World2D (
            const Passport & passport_,
            const Parallel & parallel_ )
```

Конструктор

Аргументы

in	passport↩ _	константная ссылка на паспорт расчета
in	parallel↩ _	коенстантная ссылка на параметры исполнения задачи в параллельном MPI-режиме

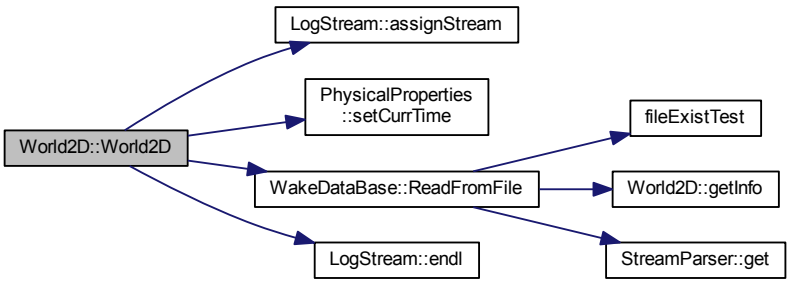
См. определение в файле World2D.cpp строка 42

```
42                                                  :
43    passport(passport_),
44    parallel(parallel_),
45    wake(*this),
46    source(*this),
47    timestat(*this),
48    cuda(Gpu(*this))
49 {
50    if (parallel.myidWork == 0)
51    {
52       std::stringstream ss;
```

```
        ss << "#" << passport.problemNumber << " (" << passport.problemName << ")";
        info.assignStream(defaults::defaultWorld2DLogStream, ss.str());
    }

    passport.physicalProperties.setCurrTime(passport.timeDiscretizationProperties.timeStart);
    currentStep = 0;

    // загрузка пелены из файла
    if (passport.wakeDiscretizationProperties.fileWake != "")
        wake.ReadFromFile(passport.wakesDir, passport.wakeDiscretizationProperties.fileWake); //Считываем из каталога с пеленой

    // загрузка положений источников из файла
    if (passport.wakeDiscretizationProperties.fileSource != "")
        source.ReadFromFile(passport.dir, passport.wakeDiscretizationProperties.fileSource); //Считываем из текущего каталога

    //airfoil.resize(GetPassport().airfoilParams.size());
    //boundary.resize(GetPassport().airfoilParams.size());
    //mechanics.resize(GetPassport().airfoilParams.size());

    switch (passport.numericalSchemes.velocityComputation)
    {
    case 0:
        velocity.reset(new VelocityBiotSavart(*this));
        break;
    case 1:
        /*
        velocity.reset(new VelocityFourier(*this));
        */
        info('e') << "VelocityFourier is not implemented now! " << std::endl;
        exit(1);
        break;
    }

    //считываем массив точек для подсчета и вывода поля скоростей и давлений
    if (passport.timeDiscretizationProperties.saveVP == 0)
        measureVP.reset(nullptr);
    else
    {
        measureVP.reset(new MeasureVP(*this));
        measureVP->ReadPointsFromFile(passport.dir);
    }

    velocity->virtualVortexesParams.resize(passport.airfoilParams.size());

    for (size_t i = 0; i < passport.airfoilParams.size(); ++i)
    {
        switch (passport.airfoilParams[i].panelsType)
        {
        case 0:
            airfoil.emplace_back(new AirfoilRect(*this, i));
            break;
        case 1:
            /*
            airfoil.emplace_back(new AirfoilCurv(*this, i));
            */
            info('e') << "AirfoilCurv is not implemented now! " << std::endl;
            exit(1);
            break;
        }

        airfoil[i]->ReadFromFile(passport.airfoilsDir);   //Считываем из каталога с коллекцией профилей

        switch (passport.airfoilParams[i].boundaryCondition)
        {
        case 0:
            /*
            boundary.emplace_back(new BoundaryMDV(*this, i));
            */
            info('e') << "BoundaryMDV is not implemented now! " << std::endl;
            exit(1);
            break;

        case 1:
```

```
            /*
            boundary.emplace_back(new BoundaryVortColl(*this, i));
            */
            info('e') << "BoundaryVortColl is not implemented now! " << std::endl;
            exit(1);
            break;

        case 2:
            boundary.emplace_back(new BoundaryConstLayerAver(*this, i));
            break;
        }


        switch (passport.airfoilParams[i].mechanicalSystemType)
        {
        case 0:
            mechanics.emplace_back(new MechanicsRigidImmovable(*this, i));
            break;
        case 1:
            mechanics.emplace_back(new MechanicsRigidGivenLaw(*this, i));
            break;
        case 2:
            mechanics.emplace_back(new MechanicsRigidOscillPart(*this, i))
;
            break;
        case 3:
            mechanics.emplace_back(new MechanicsRigidOscillMon(*this, i));
            break;
        case 4:
            mechanics.emplace_back(new MechanicsRigidRotateMon(*this, i));
            break;
        }

    }

    info.endl();
    info('i') << "Start solving problem " << passport.problemName << std::endl;
    info.endl();
}//World2D(...)
```

Граф вызовов:



Граф вызова функции:

10.38.2.2 ~World2D()

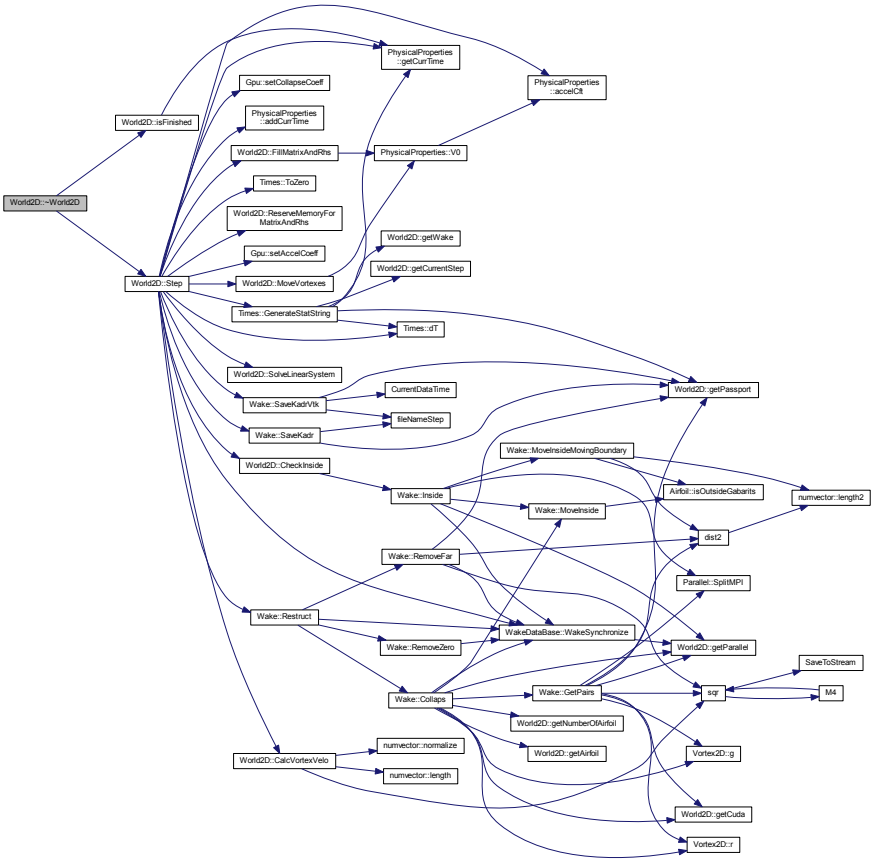
World2D::~World2D ( ) [inline]

Деструктор

См. определение в файле World2D.h строка 281

```
281 { };
```

Граф вызовов:



10.38.3 Методы

10.38.3.1 CalcVortexVelo()

```
void World2D::CalcVortexVelo (
            double dt,
            timePeriod & convTime,
            timePeriod & diffTime )
```

Вычисляем скорости вихрей (в пелене и виртуальных)

Вызывается в Step()

Аргументы

in	dt	шаг по времени
out	convTime	ссылка на промежуток времени для вычисления конвективных скоростей — пару чисел (время начала и время конца операции)
out	diffTime	ссылка на промежуток времени для вычисления диффузионных скоростей — пару чисел (время начала и время конца операции)

См. определение в файле World2D.cpp строка 771

```
{
    double tTemp[10];

    tTemp[0] = omp_get_wtime();

    if (parallel.myidWork == 0)
    {
        //Обнуляем все скорости
        convTime.first += omp_get_wtime();
        velocity->wakeVortexesParams.convVelo.clear();
        velocity->wakeVortexesParams.convVelo.resize(wake.vtx.size(), { 0.0, 0.0 });
        convTime.second += omp_get_wtime();

        diffTime.first += omp_get_wtime();
        velocity->wakeVortexesParams.I0.clear();
        velocity->wakeVortexesParams.I0.resize(wake.vtx.size(), 0.0);

        velocity->wakeVortexesParams.I1.clear();
        velocity->wakeVortexesParams.I1.resize(wake.vtx.size(), 0.0);

        velocity->wakeVortexesParams.I2.clear();
        velocity->wakeVortexesParams.I2.resize(wake.vtx.size(), { 0.0, 0.0 });

        velocity->wakeVortexesParams.I3.clear();
        velocity->wakeVortexesParams.I3.resize(wake.vtx.size(), { 0.0, 0.0 });

        velocity->wakeVortexesParams.diffVelo.clear();
        velocity->wakeVortexesParams.diffVelo.resize(wake.vtx.size(), { 0.0, 0.0 });

        velocity->wakeVortexesParams.epsastWake.clear();
        velocity->wakeVortexesParams.epsastWake.resize(wake.vtx.size(), 0.0);
        diffTime.second += omp_get_wtime();

        //Создаем массивы под виртуальные вихри
        convTime.first += omp_get_wtime();
        velocity->virtualVortexesParams.clear();
        velocity->virtualVortexesParams.resize(boundary.size());
        convTime.second += omp_get_wtime();

        for (size_t bou = 0; bou < boundary.size(); ++bou)
        {
            convTime.first += omp_get_wtime();
            velocity->virtualVortexesParams[bou].convVelo.clear();
            velocity->virtualVortexesParams[bou].convVelo.resize(
boundary[bou]->virtualWake.vtx.size(), { 0.0, 0.0 });
            convTime.second += omp_get_wtime();

            diffTime.first += omp_get_wtime();
            velocity->virtualVortexesParams[bou].I0.clear();
            velocity->virtualVortexesParams[bou].I0.resize(boundary[bou]->virtualWake.vtx.
size(), 0.0);

            velocity->virtualVortexesParams[bou].I1.clear();
            velocity->virtualVortexesParams[bou].I1.resize(boundary[bou]->virtualWake.vtx.
size(), 0.0);

            velocity->virtualVortexesParams[bou].I2.clear();
            velocity->virtualVortexesParams[bou].I2.resize(boundary[bou]->virtualWake.vtx.
size(), { 0.0, 0.0 });

            velocity->virtualVortexesParams[bou].I3.clear();
            velocity->virtualVortexesParams[bou].I3.resize(boundary[bou]->virtualWake.vtx.
size(), { 0.0, 0.0 });

            velocity->virtualVortexesParams[bou].diffVelo.clear();
            velocity->virtualVortexesParams[bou].diffVelo.resize(
boundary[bou]->virtualWake.vtx.size(), { 0.0, 0.0 });

            velocity->virtualVortexesParams[bou].epsastWake.clear();
```

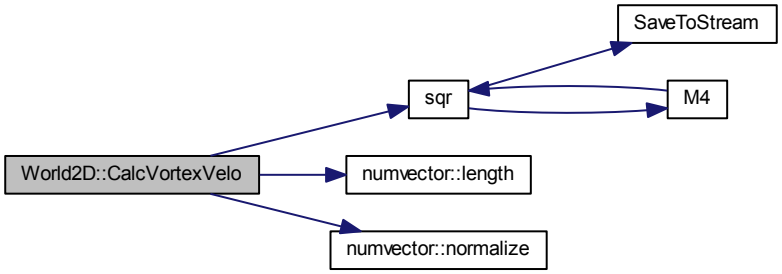
```
            velocity->virtualVortexesParams[bou].epsastWake.resize(boundary[bou]->virtualWake.vtx.size(), 0.0);
            diffTime.second += omp_get_wtime();
        }
    }



    tTemp[1] = omp_get_wtime();

    /*
    info('t') << "st=1: "
              << boundary[0]->virtualWake.vtx[0].r() << " "
              << velocity->virtualVortexesParams[0].diffVelo[0] << " "
              << velocity->virtualVortexesParams[0].convVelo[0] << " "
              << GetPassport().physicalProperties.V0() << std::endl;
    */

    convTime.first += omp_get_wtime();

    //Конвективные скорости всех вихрей (и в пелене, и виртуальных), индуцируемые вихрями в пелене
    //Подготовка CUDA

    //double timeStartRefresh = omp_get_wtime();

#if defined(__CUDACC__) || defined(USE_CUDA)
    cuda.RefreshWake();
    cuda.RefreshAfls();
#endif

    //double timeEndRefresh = omp_get_wtime();

    //info('t') << "Refresh_time = " << timeEndRefresh - timeStartRefresh << std::endl;

    //Скорости всех вихрей (и в пелене, и виртуальных), индуцируемые вихрями в пелене
    velocity->CalcConvVelo();

    tTemp[2] = omp_get_wtime();

    /*
    info('t') << "st=2: "
              << boundary[0]->virtualWake.vtx[0].r() << " "
              << velocity->virtualVortexesParams[0].diffVelo[0] << " "
              << velocity->virtualVortexesParams[0].convVelo[0] << " "
              << GetPassport().physicalProperties.V0() << std::endl;
    */

    //std::ostringstream sss;
    //sss << "epsast";
    //std::ofstream epsastFile(sss.str());
    //for (size_t i = 0; i < velocity->wakeVortexesParams.epsastWake.size(); ++i)
    //  epsastFile << velocity->wakeVortexesParams.epsastWake[i] << std::endl;
    //epsastFile.close();


    //POLARA (убрали вычисление конвективных скоростей по закону Био-Савара от слоев)
    //for (size_t bou = 0; bou < boundary.size(); ++bou)
    //
    //  boundary[bou]->GetConvVelocityToSetOfPoints(wake.vtx, velocity->wakeVortexesParams.convVelo);
    //
    //  for (size_t targetBou = 0; targetBou < boundary.size(); ++targetBou)
    //      boundary[bou]->GetConvVelocityToSetOfPoints(boundary[targetBou]->virtualWake, velocity->virtualVortexesParams[targetBou].convVelo);
    //}

    //POLARA
    //вычисление конвективных скоростей по закону Био-Савара от виртуальных вихрей
    for (size_t bou = 0; bou < boundary.size(); ++bou)
    {

        //Влияние на пелену от виртуальных вихрей
#if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_CONVVIRT))
        boundary[bou]->GPUGetConvVelocityToSetOfPointsFromVirtualVortexes(wake, velocity->wakeVortexesParams.convVelo);
#else
        boundary[bou]->GetConvVelocityToSetOfPointsFromVirtualVortexes(wake, velocity->wakeVortexesParams.convVelo);
#endif

        //Скорости самих виртуальных вихрей
        boundary[bou]->GetConvVelocityAtVirtualVortexes(velocity->virtualVortexesParams[bou].convVelo);

    }

    tTemp[3] = omp_get_wtime();

```

```
    convTime.second += omp_get_wtime();

    /*
    info('t') << "st=3: "
              << boundary[0]->virtualWake.vtx[0].r() << " "
              << velocity->virtualVortexesParams[0].diffVelo[0] << " "
              << velocity->virtualVortexesParams[0].convVelo[0] << " "
              << GetPassport().physicalProperties.V0() << std::endl;
    */

    //std::ostringstream sss;
    //sss << "EpsAst_";
    //std::ofstream bouVirtVeloFile(sss.str());
    //for (size_t i = 0; i < velocity->virtualVortexesParams[0].convVelo.size(); ++i)
    //  bouVirtVeloFile << velocity->virtualVortexesParams[0].convVelo[i][0] << " " << velocity->virtualVortexesParams[0].convVelo[i][1] \
    //  << " " << velocity->virtualVortexesParams[0].epsastWake[i] << std::endl;
    //bouVirtVeloFile.close();

    //exit(1);

    //std::ostringstream sss;
    //sss << "bouVirtVelo_";
    //std::ofstream bouVirtVeloFile(sss.str());
    //for (size_t i = 0; i < velocity->virtualVortexesParams[0].convVelo.size(); ++i)
    //  bouVirtVeloFile << velocity->virtualVortexesParams[0].convVelo[i][0] << " " << velocity->virtualVortexesParams[0].convVelo[i][1] << std::endl;
    //bouVirtVeloFile.close();

    //std::ostringstream ss;
    //ss << "bouVelo_";
    //std::ofstream bouVeloFile(ss.str());
    //bouVeloFile << velocity->wakeVortexesParams.convVelo << std::endl;
    //bouVeloFile.close();


    diffTime.first += omp_get_wtime();

    //Вычисление диффузионных скоростей вихрей в следе и виртуальных вихрей, обусловленных влиянием вихрей (тех, что в следе, и виртуальных вихрей)
    velocity->CalcDiffVelo();

    tTemp[4] = omp_get_wtime();

    /*
    info('t') << "st=4: "
          << boundary[0]->virtualWake.vtx[0].r() << " "
          << velocity->virtualVortexesParams[0].diffVelo[0] << " "
          << velocity->virtualVortexesParams[0].convVelo[0] << " "
          << GetPassport().physicalProperties.V0() << std::endl;
    */

    //Вычисление диффузионных скоростей вихрей в следе и виртуальных вихрей, обусловленных влиянием поверхностей


    for (size_t afl = 0; afl < airfoil.size(); ++afl)
    {
#if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_I0I3))
        airfoil[afl]->GPUGetDiffVelocityI0I3ToSetOfPointsAndViscousStresses(wake, velocity->wakeVortexesParams.epsastWake, velocity->wakeVortexesParams.I0, velocity->wakeVortexesParams.I3);
#else
        airfoil[afl]->GetDiffVelocityI0I3ToSetOfPointsAndViscousStresses(wake, velocity->wakeVortexesParams.epsastWake, velocity->wakeVortexesParams.I0, velocity->wakeVortexesParams.I3);
#endif

        for (size_t bou = 0; bou < boundary.size(); ++bou)
#if (defined(__CUDACC__) || defined(USE_CUDA)) && (defined(CU_I0I3))
            airfoil[afl]->GPUGetDiffVelocityI0I3ToSetOfPointsAndViscousStresses(boundary[bou]->virtualWake, velocity->virtualVortexesParams[bou].epsastWake, velocity->virtualVortexesParams[bou].I0, velocity->virtualVortexesParams[bou].I3);
#else
            airfoil[afl]->GetDiffVelocityI0I3ToSetOfPointsAndViscousStresses(boundary[bou]->virtualWake, velocity->virtualVortexesParams[bou].epsastWake, velocity->virtualVortexesParams[bou].I0, velocity->virtualVortexesParams[bou].I3);
#endif
    }

    tTemp[5] = omp_get_wtime();

    //влияние поверхности
    Point2D I3;
    double I0;

    for (size_t vt = 0; vt < velocity->wakeVortexesParams.diffVelo.size(); ++vt)
```

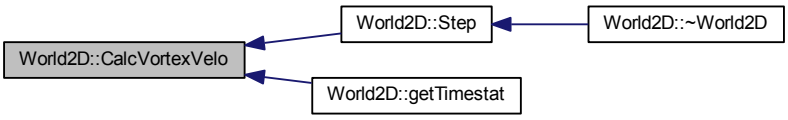
```
    {
        velocity->wakeVortexesParams.I0[vt] *= velocity->wakeVortexesParams.epsastWake[vt];
        velocity->wakeVortexesParams.I0[vt] += DPI * sqr(velocity->wakeVortexesParams.epsastWake[vt]);

        I3 = velocity->wakeVortexesParams.I3[vt];
        I0 = velocity->wakeVortexesParams.I0[vt];

        if (fabs(I0) > 1.e-8)
            velocity->wakeVortexesParams.diffVelo[vt] += I3 * (1.0 / I0);
    }

    tTemp[6] = omp_get_wtime();

    for (size_t targetBou = 0; targetBou < boundary.size(); ++targetBou)
    for (size_t vt = 0; vt < velocity->virtualVortexesParams[targetBou].diffVelo.size(); ++vt)
    {
        velocity->virtualVortexesParams[targetBou].I0[vt] *= velocity->virtualVortexesParams[targetBou].epsastWake[vt];
        velocity->virtualVortexesParams[targetBou].I0[vt] += DPI * sqr(velocity->virtualVortexesParams[targetBou].epsastWake[vt]);

        I3 = velocity->virtualVortexesParams[targetBou].I3[vt];
        I0 = velocity->virtualVortexesParams[targetBou].I0[vt];

        if (fabs(I0) > 1.e-8)
            velocity->virtualVortexesParams[targetBou].diffVelo[vt] += I3 * (1.0 / I0);
    }

    tTemp[7] = omp_get_wtime();

    diffTime.second += omp_get_wtime();

    /*
    info('t') << "st=5: "
          << boundary[0]->virtualWake.vtx[0].r() << " "
          << velocity->virtualVortexesParams[0].diffVelo[0] << " "
          << velocity->virtualVortexesParams[0].convVelo[0] << " "
          << GetPassport().physicalProperties.V0() << std::endl;
    */


    //std::ofstream stressFile("Stresses", std::ios::app);
    //stressFile << currentStep << "    " << GetPassport().physicalProperties.getCurrTime() << "    ";
    //for (size_t i = 0; i < airfoil[0]->viscousStress.size(); ++i)
    //  stressFile << airfoil[0]->viscousStress[i] << " ";
    //stressFile << "\n";
    //stressFile.close();


    diffTime.first += omp_get_wtime();

    //контроль застрелов диффузионной скорости
    if (parallel.myidWork == 0)
    for (size_t i = 0; i < velocity->wakeVortexesParams.diffVelo.size(); ++i)
    {
        Point2D& diffV = velocity->wakeVortexesParams.diffVelo[i];

        diffV *= passport.physicalProperties.nu;

        if (diffV.length() > 0.5*passport.physicalProperties.vRef)
            diffV.normalize(0.5*passport.physicalProperties.vRef);
    }

    /*
    info('t') << "st=6: "
              << boundary[0]->virtualWake.vtx[0].r() << " "
              << velocity->virtualVortexesParams[0].diffVelo[0] << " "
              << velocity->virtualVortexesParams[0].convVelo[0] << " "
              << GetPassport().physicalProperties.V0() << std::endl;
    */

    //контроль "застрелов" диффузионной скорости
    if (parallel.myidWork == 0)
    for (size_t bou = 0; bou < velocity->virtualVortexesParams.size();  ++bou)
    for (size_t i = 0; i < velocity->virtualVortexesParams[bou].diffVelo.size(); ++i)
    {
        Point2D& diffV = velocity->virtualVortexesParams[bou].diffVelo[i];

        diffV *= passport.physicalProperties.nu;
        if (diffV.length() > 0.5*passport.physicalProperties.vRef)
            diffV.normalize(0.5*passport.physicalProperties.vRef);
    }
```

```
    for (size_t i = 0; i < airfoil[0]->viscousStress.size(); ++i)
        airfoil[0]->viscousStress[i] *= passport.physicalProperties.nu;

    diffTime.second += omp_get_wtime();


    tTemp[8] = omp_get_wtime();

    /*
    info('t') << "st=7: "
              << boundary[0]->virtualWake.vtx[0].r() << " "
              << velocity->virtualVortexesParams[0].diffVelo[0] << " "
              << velocity->virtualVortexesParams[0].convVelo[0] << " "
              << GetPassport().physicalProperties.V0() << std::endl;
    */

    //std::ostringstream sss;
    //sss << "velo_";
    //std::ofstream veloFile(sss.str());
    //for (int i = 0; i < velocity->diffVirtualVelo[0].size(); ++i)
    //  veloFile << velocity->diffVirtualVelo[0][i] << std::endl;
    //veloFile.close();

    tTemp[9] = omp_get_wtime();

#pragma warning (push)
#pragma warning (disable: 4010)
    //info('t') << "Times: " << \
        tTemp[1] - tTemp[0] << " " << \
        tTemp[2] - tTemp[1] << " " << \
        tTemp[3] - tTemp[2] << " " << \
        tTemp[4] - tTemp[3] << " " << \
        tTemp[5] - tTemp[4] << " " << \
        tTemp[6] - tTemp[5] << " " << \
        tTemp[7] - tTemp[6] << " " << \
        tTemp[8] - tTemp[7] << " " << \
        tTemp[9] - tTemp[8] << " " << \
        std::endl;
#pragma warning (pop)

}//CalcVortexVelo(...)
```

Граф вызовов:



Граф вызова функции:



10.38.3.2 CheckInside()

```
void World2D::CheckInside (
            std::vector< Point2D > & newPos,
            timePeriod & time )
```

Проверка проникновения вихрей внутрь профиля

Вызывается в Step()

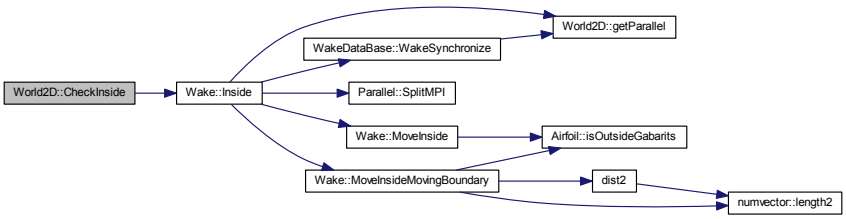
Аргументы

in	newPos	новые позиции вихрей
out	time	ссылка на промежуток времени — пару чисел (время начала и время конца операции)

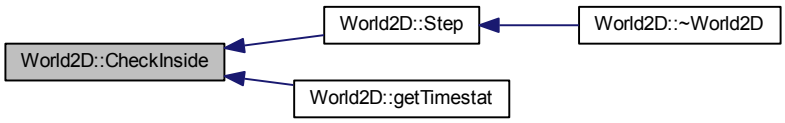
См. определение в файле World2D.cpp строка 467

```
468 {
469     time.first = omp_get_wtime();
470
471     for (size_t afl = 0; afl < airfoil.size(); ++afl)
472         wake.Inside(newPos, *airfoil[afl], mechanics[afl]->isMoves, *
    oldAirfoil[afl]);
473
474     time.second = omp_get_wtime();
475 }
```

Граф вызовов:



Граф вызова функции:



10.38.3.3 FillMatrixAndRhs()

void World2D::FillMatrixAndRhs (
timePeriod & time)

Заполнение матрицы системы для всех профилей

Вызывается в Step()

Аргументы

out	time	ссылка на промежуток времени — пару чисел (время начала и время конца операции)

Необходимо сделать Влияние присоединенных слоев от других профилей пока выключено (для поступательного движения вроде не нужно)

См. определение в файле World2D.cpp строка 533

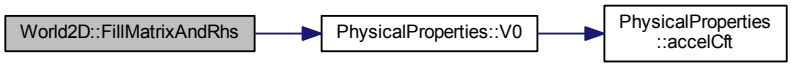
```
534 {
535     time.first = omp_get_wtime();
536
537     Eigen::MatrixXd locMatr;
538     Eigen::MatrixXd otherMatr;
539     Eigen::VectorXd locLastLine, locLastCol;
540     Eigen::VectorXd locRhs;
541     std::vector <Eigen::MatrixXd> mechRows; //строки с уравнениями для аэроупругой системы (слагаемые,
    отвечающие за интенсивность вихревого слоя, в механическом уравнении)
542     std::vector <Eigen::MatrixXd> mechCols; //столбцы с уравнениями для аэроупругой системы (слагаемые,
    отвечающие за скорость, в граничном условии)
543     std::vector <Eigen::MatrixXd> mechCross; //пересечение строк и столбцов, относящхся к механической
    системе (слагаемое, отвечаюшее за скорость, в механическом уравнении)
544     std::vector <std::vector<double>> mechRhs;  //компоненты правой части уравнений для аэроупругой системы
545
546
547     double locLastRhs = 0.0;
548
549     //обнуляем матрицу на первом шаге расчета
550     if (currentStep == 0)
551     {
552         for (int i = 0; i < matr.rows(); ++i)
553         for (int j = 0; j < matr.cols(); ++j)
554             matr(i, j) = 0.0;
555     }
556
557     mechRows.resize(mechanics.size());
558     mechCols.resize(mechanics.size());
559     mechCross.resize(mechanics.size());
```

```
    mechRhs.resize(mechanics.size());

    for (size_t mech = 0; mech < boundary.size(); ++mech)
    {
        mechRows[mech].resize(mechanics[mech]->degOfFreedom, boundary[mech]->GetUnknownsSize());
        for (int i = 0; i < mechRows[mech].rows(); ++i)
        for (int j = 0; j < mechRows[mech].cols(); ++j)
            mechRows[mech](i, j) = 0.0;

        mechCols[mech].resize(boundary[mech]->GetUnknownsSize(), mechanics[mech]->degOfFreedom);
        for (int i = 0; i < mechCols[mech].rows(); ++i)
        for (int j = 0; j < mechCols[mech].cols(); ++j)
            mechCols[mech](i, j) = 0.0;

        mechCross[mech].resize(mechanics[mech]->degOfFreedom, mechanics[mech]->degOfFreedom);
        for (int i = 0; i < mechCross[mech].rows(); ++i)
        for (int j = 0; j < mechCross[mech].cols(); ++j)
            mechCross[mech](i, j) = 0.0;

        mechRhs[mech].resize(mechanics[mech]->degOfFreedom);
    }

    size_t currentRow = 0;

    for (size_t bou = 0; bou < boundary.size(); ++bou)
    {
        size_t nVars;

        if (parallel.myidWork == 0)
        {
            nVars = boundary[bou]->GetUnknownsSize();

            if (currentStep == 0)
            {
                locMatr.resize(nVars, nVars);
                locLastLine.resize(nVars);
                locLastCol.resize(nVars);
            }
            locRhs.resize(nVars);
        }

        if (currentStep == 0 || mechanics[bou]->isDeform)
        {
            if (parallel.myidWork == 0)
            {
                boundary[bou]->FillMatrixSelf(locMatr, locLastLine, locLastCol);
                mechanics[bou]->FillMechanicsRowsAndCross(mechRows[bou], mechCross[bou]);
            }
        }

        boundary[bou]->FillRhs(passport.physicalProperties.V0(), locRhs, &locLastRhs, mechanics[bou]->isMoves, mechanics[bou]->isDeform);
        mechanics[bou]->FillMechanicsRhs(mechRhs[bou]);

        Eigen::MatrixXd attCols;
        Eigen::MatrixXd attRhs;
        attCols.resize(mechCols[bou].rows(), mechCols[bou].cols());
        attRhs.resize(mechCols[bou].rows(), 1);


        for (int i = 0; i < attCols.rows(); ++i)
        {
            for (int j = 0; j < attCols.cols(); ++j)
                attCols(i, j) = 0.0;

            attRhs(i) = 0.0;
        }

        mechanics[bou]->FillAtt(attCols, attRhs);

/*      for (size_t i = 0; i < airfoil.size(); ++i)
        {
            if (i != bou)
            if (mechanics[i]->isDeform || mechanics[i]->isMoves)
                boundary[bou]->FillRhsFromOther(*airfoil[i], locRhs);
        }
*/

        //размазываем матрицу
        if (parallel.myidWork == 0)
        {
            for (int i = 0; i < nVars; ++i)
            {
                if (currentStep == 0 || mechanics[bou]->isDeform)
```

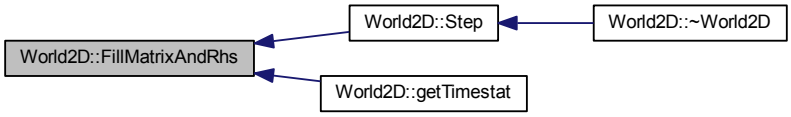
```
            {
                for (int j = 0; j < nVars; ++j)
                    matr(i + currentRow, j + currentRow) = locMatr(i, j);
                matr(currentRow + nVars, i + currentRow) = locLastLine(i);
                matr(i + currentRow, currentRow + nVars) = locLastCol(i);
            }

            rhs(i + currentRow) = locRhs(i);
        }
        rhs(currentRow + nVars) = locLastRhs;

        //размазываем механическую систему
        for (int i = 0; i < mechanics[bou]->degOfFreedom; ++i)
        {
            //\todo пока матрица считается только на первом шаге
            if (currentStep == 0)
            {
                for (int j = 0; j < nVars; ++j)//строки и столбцы
                {
                    matr(currentRow + nVars + 1 + i, currentRow + j) = mechRows[bou](i, j);
                    matr(currentRow + j, currentRow + nVars + 1 + i) = mechCols[bou](j, i);
                }

                for (int j = 0; j < mechanics[bou]->degOfFreedom; ++j)//пересечения строк и столбцов
                {
                    matr(currentRow + nVars + 1 + i, currentRow + nVars + 1 + j) = mechCross[bou](i, j);
                }
            }
            rhs(currentRow + nVars + 1 + i) = mechRhs[bou][i];
        }


        //добавляем влияние присоединенных слоев
        for (int i = 0; i < nVars; ++i)
        {
            //\todo пока матрица считается только на первом шаге
            if (currentStep == 0)
            {
                for (int j = 0; j < mechanics[bou]->degOfFreedom; ++j) //строки и столбцы
                    matr(currentRow + i, currentRow + nVars + 1 + j) += attCols(i, j);
            }

            rhs(currentRow + i) += attRhs(i);
        }


        //\todo пока матрица считается только на первом шаге
        if (currentStep == 0)
        {
            size_t currentCol = 0;
            for (size_t oth = 0; oth < boundary.size(); ++oth)
            {
                size_t nVarsOther = boundary[oth]->GetUnknownsSize();

                if (bou != oth)
                {
                    otherMatr.resize(nVars, nVarsOther);

                    boundary[bou]->FillMatrixFromOther(*boundary[oth], otherMatr);

                    //размазываем матрицу
                    for (size_t i = 0; i < nVars; ++i)
                    {
                        for (size_t j = 0; j < nVarsOther; ++j)
                            matr(i + currentRow, j + currentCol) = otherMatr(i, j);
                    }
                }// if (bou != oth)
                currentCol += nVarsOther + 1 + mechanics[oth]->degOfFreedom;
            }// for oth
        }// if (currentStep == 0 || mechanics[oth]->isMoves)

        currentRow += nVars + 1 + mechanics[bou]->degOfFreedom;

    }// if (parallel.myidWork == 0)
}// for bou


/*
std::ostringstream ss;
ss << "matrix_";
ss << currentStep;
std::ofstream matrFile(ss.str());
SaveToStream(matr, matrFile);
matrFile.close();

```

```
729
730
731     std::ostringstream sss;
732     sss << "rhs_";
733     sss << currentStep;
734     std::ofstream rhsFile(sss.str());
735     SaveToStream(rhs, rhsFile);
736     rhsFile.close();
737     */
738
739
740     time.second = omp_get_wtime();
741 }//FillMatrixAndRhs(...)
```

Граф вызовов:



Граф вызова функции:



10.38.3.4 GenerateMechanicsHeader()

void World2D::GenerateMechanicsHeader (
size_t mechanicsNumber) [inline]

Метод-обертка для вызова метода генерации заголовка файла нагрузок и заголовка файла положения (последнее — если профиль движется)

Аргументы

in	mechanicsNumber	номер профиля, для которого генерируется заголовок файла

См. определение в файле World2D.h строка 293

```
294     {
295         mechanics[mechanicsNumber]->GenerateForcesHeader();
296         mechanics[mechanicsNumber]->GeneratePositionHeader();
297     }
```

10.38.3.5 getAirfoil()

```
const Airfoil& World2D::getAirfoil (
            size_t i ) const   [inline]
```

Возврат константной ссылки на объект профиля

Аргументы

in	i	номер профиля, константная ссылка на который возвращается

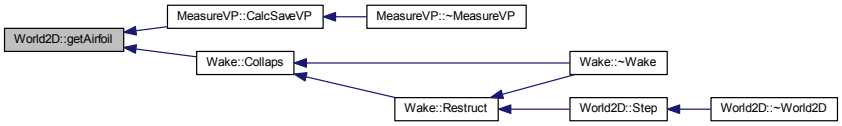
Возвращает

константную ссылку на i-й профиль

См. определение в файле World2D.h строка 158

```
158 { return *airfoil[i]; };
```

Граф вызова функции:



10.38.3.6 getBoundary()

```
const Boundary& World2D::getBoundary (
            size_t i ) const   [inline]
```

Возврат константной ссылки на объект граничного условия

Аргументы

in	i	номер граничного условия, константная ссылка на которое возвращается

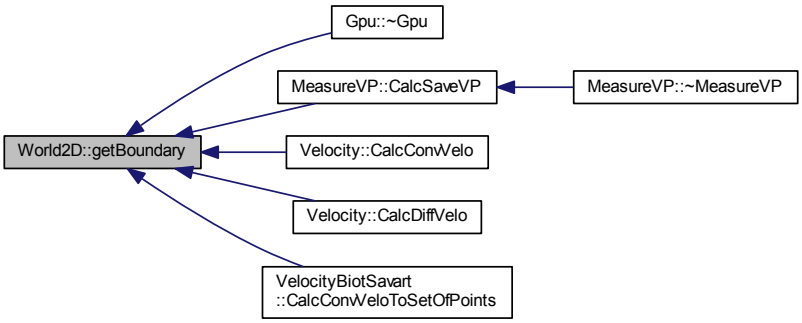
Возвращает

константную ссылку на i-е граничное условие

См. определение в файле World2D.h строка 175

175 { return *boundary[i]; };

Граф вызова функции:



10.38.3.7 getCuda()

const Gpu& World2D::getCuda ( ) const [inline]

Возврат константной ссылки на объект, связанный с видеокартой (GPU)

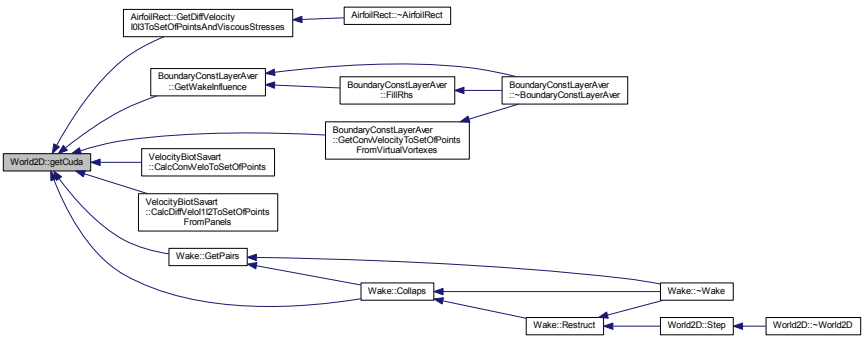
Возвращает

константную ссылку на объект, связанный с видеокартой (GPU)

См. определение в файле World2D.h строка 221

221 { return cuda; };

Граф вызова функции:



10.38.3.8 getCurrentStep()

size_t World2D::getCurrentStep ( ) const [inline]

Возврат номера текущего временного шага

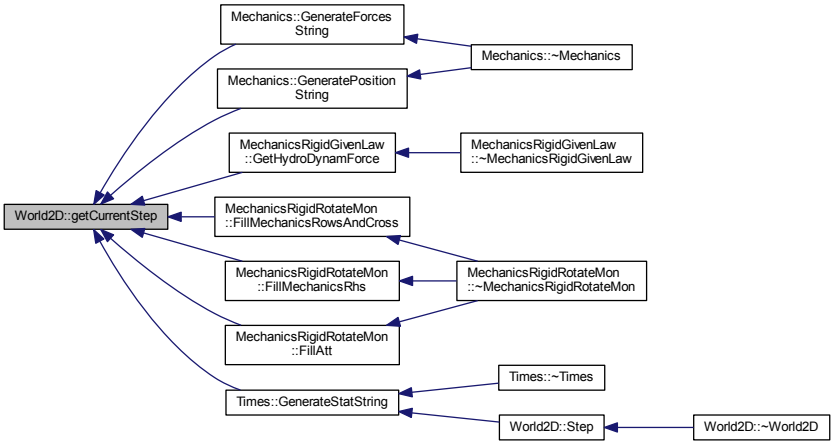
Возвращает

номера текущего временного шага

См. определение в файле World2D.h строка 226

```
226 { return currentStep; };
```

Граф вызова функции:



10.38.3.9 getInfo() [1/2]

LogStream& World2D::getInfo ( ) const [inline]

Возврат ссылки на объект LogStream Используется в техничеких целях для организации вывода

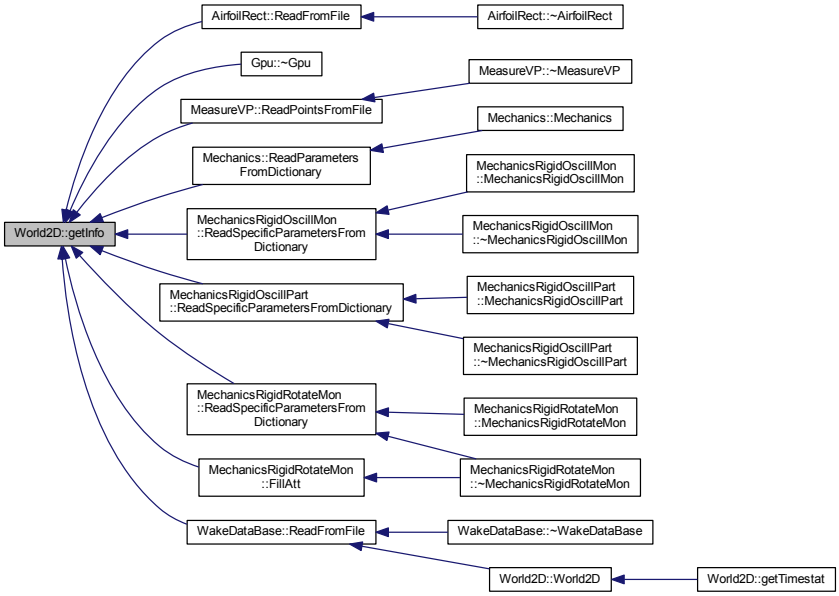
Возвращает

ссылку на объект LogStream

См. определение в файле World2D.h строка 145

```
145 { return info; };
```

Граф вызова функции:



10.38.3.10 getInfo() [2/2]

```
std::ostream& World2D::getInfo (
            char x ) const    [inline]
```

Возврат ссылки на поток вывода информации Необходимо для вывода телеметрической информации, информации об ошибках и т.п.

Аргументы

in	x	символ, определяющий стиль вывода сообщения

Возвращает

ссылку на поток вывода информации

См. определение в файле World2D.h строка 152

```
152 { return info(x); };
```

10.38.3.11 getNonConstAirfoil()

```
Airfoil& World2D::getNonConstAirfoil (
            size_t i ) const   [inline]
```

Возврат неконстантной ссылки на объект профиля

Аргументы

in	i	номер профиля, неконстантная ссылка на который возвращается

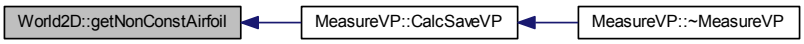
Возвращает

неконстантную ссылку на i-й профиль

См. определение в файле World2D.h строка 164

```
164 { return *airfoil[i]; };
```

Граф вызова функции:



10.38.3.12 getNonConstBoundary()

```
Boundary& World2D::getNonConstBoundary (
            size_t i ) const   [inline]
```

Возврат неконстантной ссылки на объект граничного условия

Аргументы

in	i	номер граничного условия, неконстантная ссылка на которое возвращается

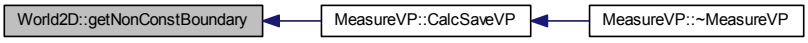
Возвращает

неконстантную ссылку на i-е граничное условие

См. определение в файле World2D.h строка 181

```
181 { return *boundary[i]; };
```

Граф вызова функции:



10.38.3.13 getNonConstVelocity()

Velocity& World2D::getNonConstVelocity ( ) const [inline]

Возврат неконстантной ссылки на объект для вычисления скоростей

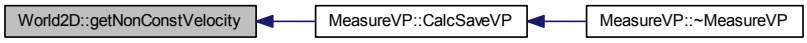
Возвращает

неконстантную ссылку на объект для вычисления скоростей

См. определение в файле World2D.h строка 206

```
206 { return *velocity; };
```

Граф вызова функции:



10.38.3.14 getNumberOfAirfoil()

size_t World2D::getNumberOfAirfoil ( ) const [inline]

Возврат количества профилей в задаче

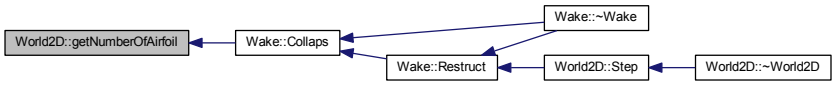
Возвращает

количество профилей в задаче

См. определение в файле World2D.h строка 169

```
169 { return airfoil.size(); };
```

Граф вызова функции:



10.38.3.15 getNumberOfBoundary()

size_t World2D::getNumberOfBoundary ( ) const [inline]

Возврат количества граничных условий в задаче

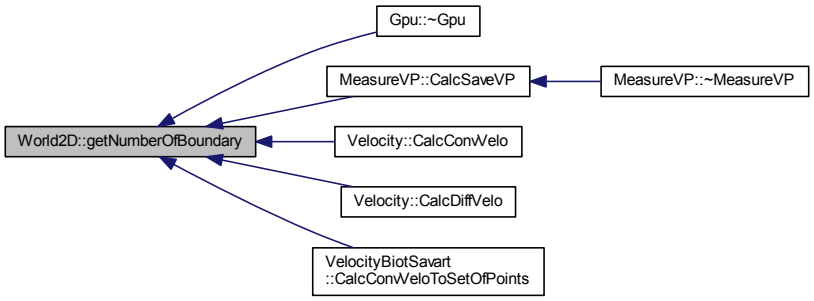
Возвращает

количество граничных условий в задаче

См. определение в файле World2D.h строка 186

```
186 { return boundary.size(); };
```

Граф вызова функции:



10.38.3.16 getParallel()

const Parallel& World2D::getParallel ( ) const [inline]

Возврат константной ссылки на параметры распараллеливания по MPI.

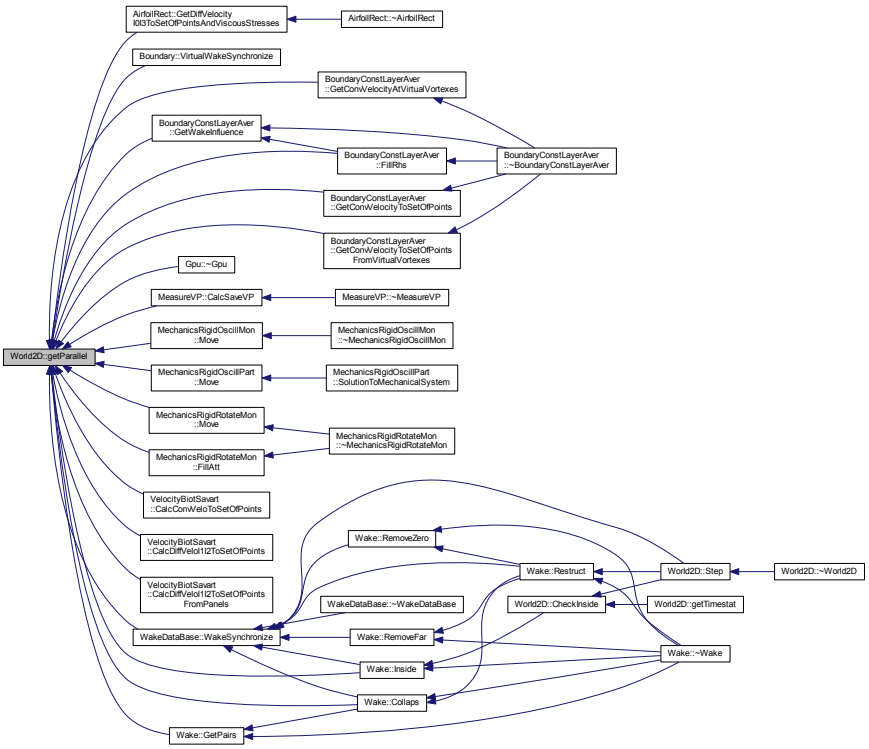
Возвращает

константную ссылку на параметры распараллеливания по MPI

См. определение в файле World2D.h строка 216

```
216 { return parallel; };
```

Граф вызова функции:



10.38.3.17 getPassport()

const Passport& World2D::getPassport ( ) const [inline]

Возврат константной ссылки на паспорт

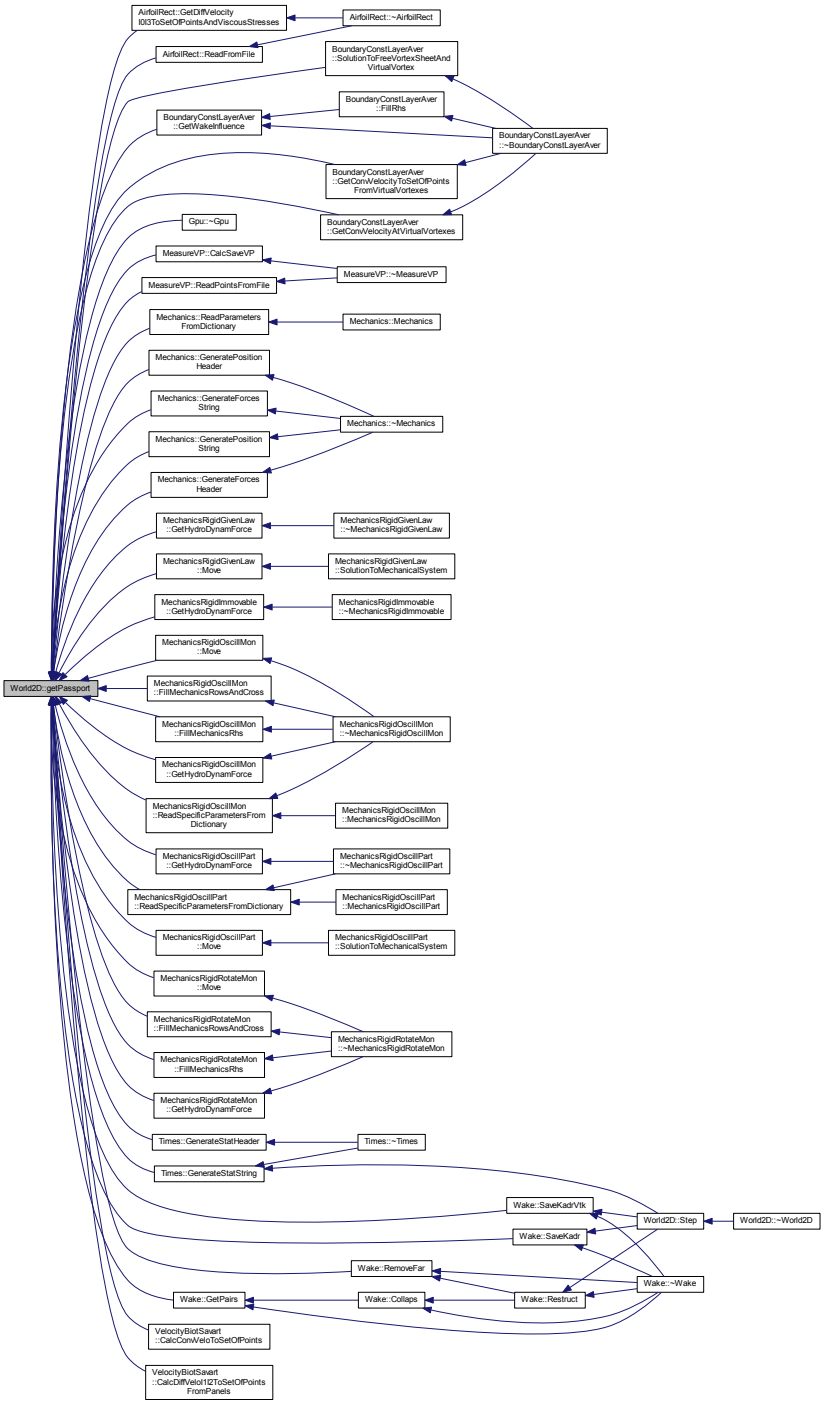
Возвращает

константную ссылку на паспорт

См. определение в файле World2D.h строка 211

```
211 { return passport; };
```

Граф вызова функции:



10.38.3.18 getSource()

const WakeDataBase& World2D::getSource ( ) const [inline]

Возврат константной ссылки на источники в области течения

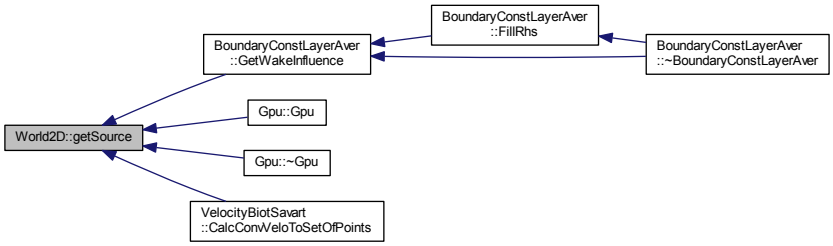
Возвращает

константную ссылку на источники в области течения

См. определение в файле World2D.h строка 196

```
196 { return source; };
```

Граф вызова функции:



10.38.3.19 getTimestat()

const Times& World2D::getTimestat ( ) const [inline]

Возврат константной ссылки на временную статистику выполнения шага расчета по времени

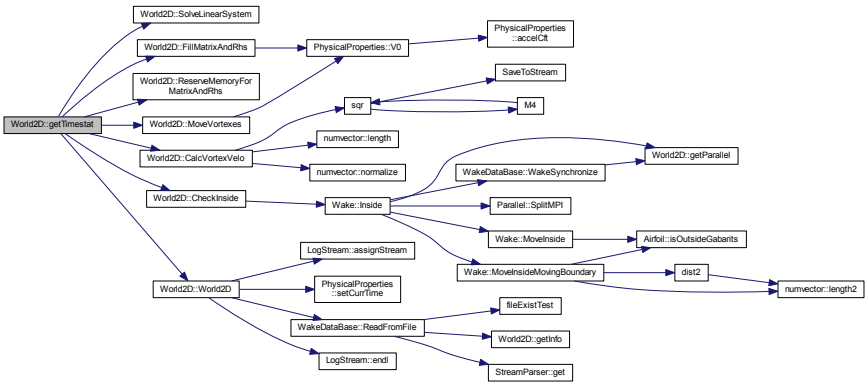
Возвращает

константную ссылку на временную статистику выполнения шага расчета по времени

См. определение в файле World2D.h строка 231

```
231 { return timestat; };
```

Граф вызовов:



10.38.3.20 getVelocity()

const Velocity& World2D::getVelocity ( ) const [inline]

Возврат константной ссылки на объект для вычисления скоростей

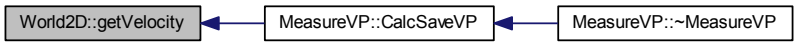
Возвращает

константную ссылку на объект для вычисления скоростей

См. определение в файле World2D.h строка 201

```
201 { return *velocity; };
```

Граф вызова функции:



10.38.3.21 getWake()

const Wake& World2D::getWake ( ) const [inline]

Возврат константной ссылки на вихревой след

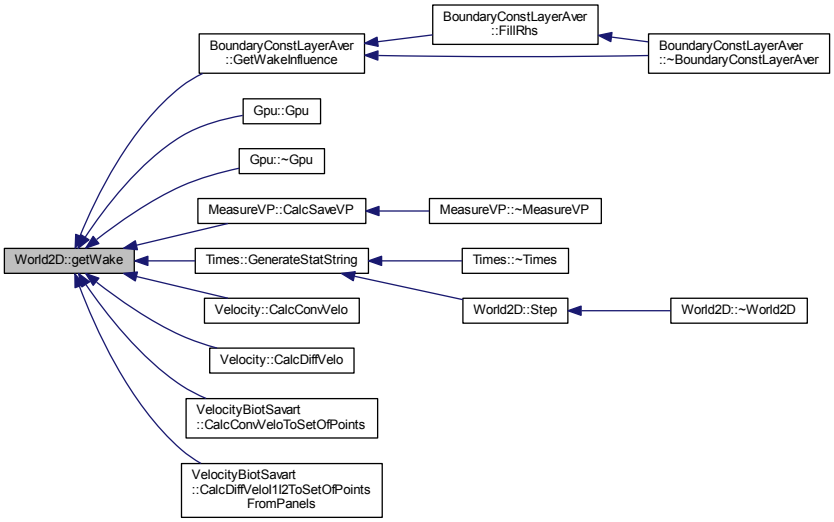
Возвращает

константную ссылку на вихревой след

См. определение в файле World2D.h строка 191

```
191 { return wake; };
```

Граф вызова функции:



10.38.3.22 isFinished()

bool World2D::isFinished ( ) const

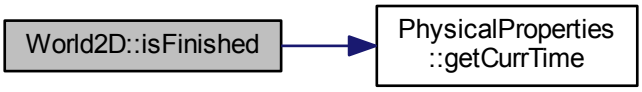
Функция, возвращающая признак завершения счета в решаемой задаче

true, если задача решена и выполнен признак останова; false если требуется еще выполнять шаги по времени

См. определение в файле World2D.cpp строка 167

```
168 {
169     return (passport.physicalProperties.getCurrTime() >=
    passport.timeDiscretizationProperties.
    timeStop);
170 }//isFinished()
```

Граф вызовов:



Граф вызова функции:

10.38.3.23 MoveVortexes()

```
void World2D::MoveVortexes (
            double dt,
            std::vector< Point2D > & newPos,
            timePeriod & time )
```

Вычисляем новые положения вихрей (в пелене и виртуальных)

Вызывается в Step()

Аргументы

in	dt	шаг по времени
out	newPos	новые позиции вихрей
out	time	ссылка на промежуток времени — пару чисел (время начала и время конца операции)

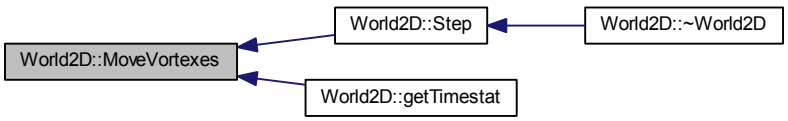
См. определение в файле World2D.cpp строка 1117

```
1118 {
1120     newPos.clear();
1121     newPos.resize(0);
1122
1123
1124     time.first = omp_get_wtime();
```

```
    if (parallel.myidWork == 0)
        for (size_t i = 0; i < wake.vtx.size(); ++i)
        {
            newPos.push_back(wake.vtx[i].r() + (velocity->wakeVortexesParams.convVelo[i] +
velocity->wakeVortexesParams.diffVelo[i] + passport.
physicalProperties.V0())*passport.
timeDiscretizationProperties.dt);
            // info('t') << *(newPos.end() - 1) << std::endl;
        }

    if (parallel.myidWork == 0)
        for (size_t bou = 0; bou < boundary.size(); ++bou)
            for (size_t i = 0; i < boundary[bou]->virtualWake.vtx.size(); ++i)
            {
                wake.vtx.push_back(boundary[bou]->virtualWake.vtx[i]);
                newPos.push_back(boundary[bou]->virtualWake.vtx[i].r() \
                    + (velocity->virtualVortexesParams[bou].diffVelo[i] +
                        velocity->virtualVortexesParams[bou].convVelo[i] +
                        passport.physicalProperties.V0())*
passport.timeDiscretizationProperties.dt);
            }

    time.second = omp_get_wtime();
}
```

Граф вызовов:

Граф вызова функции:



10.38.3.24 ReserveMemoryForMatrixAndRhs()

```
void World2D::ReserveMemoryForMatrixAndRhs (
            timePeriod & time )
```

Вычисляем размер матрицы и резервируем память под нее и под правую часть

Вызывается в Step()

Аргументы

out	time	ссылка на промежуток времени — пару чисел (время начала и время конца операции)

См. определение в файле World2D.cpp строка 744

```
745 {
746     time.first = omp_get_wtime();
747
748     if (currentStep == 0)
749     {
750         size_t matrSize = boundary.size();
751
752         for (auto it = boundary.begin(); it != boundary.end(); ++it)
753             matrSize += (*it)->GetUnknownsSize();
754
755         //добавляем уравнения (их количество совпадает с количеством степеней свободы для профилей) для
    решения механической системы
756         for (auto it = mechanics.begin(); it != mechanics.end(); ++it)
757             matrSize += (*it)->degOfFreedom;
758
759         matr.resize(matrSize, matrSize);
760         matr.setZero();
761
762         rhs.resize(matrSize);
763     }
764     rhs.setZero();
765
766     time.second = omp_get_wtime();
767 }//ReserveMemoryForMatrixAndRhs(...)
```

Граф вызова функции:

10.38.3.25 SolveLinearSystem()

```
void World2D::SolveLinearSystem (
                timePeriod & time )
```

Решение системы линейных алгебраических уравнений

Вызывается в Step()

Аргументы

out	time	ссылка на промежуток времени — пару чисел (время начала и время конца операции)

См. определение в файле World2D.cpp строка 480

```
{
    time.first = omp_get_wtime();
    //sol = matr.partialPivLu().solve(rhs);


    if (currentStep == 0)
        invMatr = matr.inverse();


/*
    if (currentStep == 0)
    {
        invMatr = matr;
        std::ifstream fileMatrix("invMatrix4x3200.txt");
        int nx, ny;
        fileMatrix >> nx;
        fileMatrix >> ny;
        for (int i = 0; i < invMatr.rows(); ++i)
        {
            for (int j = 0; j < invMatr.cols(); ++j)
            {
                fileMatrix >> invMatr(i, j);
            }
        }
        fileMatrix.close();
    }
*/

/*
    if (currentStep == 0)
    {
        std::ofstream fileMatrix("invMatrix4x3200.txt");
        fileMatrix << invMatr.rows() << " " << invMatr.cols() << std::endl;
        fileMatrix.precision(18);
        for (int i = 0; i < invMatr.rows(); ++i)
        {
            for (int j = 0; j < invMatr.cols(); ++j)
            {
                fileMatrix << invMatr(i, j) << " ";
            }
            fileMatrix << std::endl;
        }
        fileMatrix.close();
    }
*/

    sol = invMatr*rhs;

    time.second = omp_get_wtime();
}
```

Граф вызова функции:

10.38.3.26 Step()

void World2D::Step ( )

Основная функция выполнения одного шага по времени

!!!

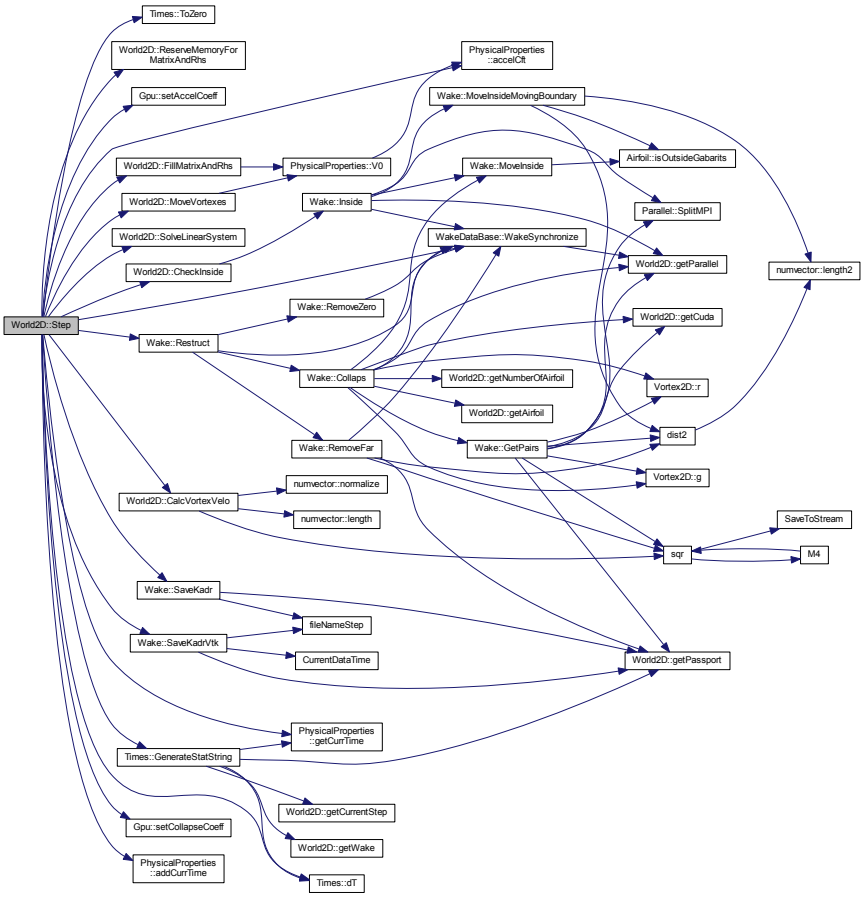
```
{

    timePeriod& time = timestat.timeWholeStep;

    //Очистка статистики
    timestat.ToZero();

    //Засечка времени в начале шага
    time.first = omp_get_wtime();

    const double& dt = passport.timeDiscretizationProperties.dt;


timestat.timeOther.first += omp_get_wtime();
    //вычисляем скорости панелей
    for (size_t i = 0; i < airfoil.size(); ++i)
        mechanics[i]->VeloOfAirfoilPanels(currentStep * dt);

    //вычисляем интенсивности присоединенных слоев
    for (size_t i = 0; i < airfoil.size(); ++i)
        boundary[i]->ComputeAttachedSheetsIntensity();

timestat.timeOther.second += omp_get_wtime();

    if (airfoil.size() > 0)
    {
        if (parallel.myidWork == 0)
            ReserveMemoryForMatrixAndRhs(timestat.timeReserveMemoryForMatrixAndRhs);

#if defined(__CUDACC__) || defined(USE_CUDA)
        cuda.setAccelCoeff(passport.physicalProperties.accelCft());
        cuda.RefreshWake();
        cuda.RefreshAfls();
#endif

        FillMatrixAndRhs(timestat.timeFillMatrixAndRhs);

        if (parallel.myidWork == 0)
            SolveLinearSystem(timestat.timeSolveLinearSystem);

        /*if (parallel.myidWork == 0)
        {
            for (int i = 0; i < sol.size(); ++i)
            {
                info('t') << i << " " << sol(i) << std::endl;
            }
        }*/


timestat.timeOther.first += omp_get_wtime();
        if (parallel.myidWork == 0)
        {
            size_t currentRow = 0;
            for (size_t bou = 0; bou < boundary.size(); ++bou)
            {
                size_t nVars = boundary[bou]->GetUnknownsSize();
                Eigen::VectorXd locSol, locMechSol;
                locSol.resize(nVars);
                locMechSol.resize(mechanics[bou]->degOfFreedom);
                for (size_t i = 0; i < nVars; ++i)
                    locSol(i) = sol(currentRow + i);

                for (size_t i = 0; i < mechanics[bou]->degOfFreedom; ++i)
                    locMechSol(i) = sol(currentRow + nVars + 1 + i);

                boundary[bou]->SolutionToFreeVortexSheetAndVirtualVortex(locSol);
                mechanics[bou]->SolutionToMechanicalSystem(locMechSol);
                currentRow += nVars + 1 + mechanics[bou]->degOfFreedom;
            }

            /*
            std::ostringstream ss;
            ss << "solution_";
            ss << currentStep;
            std::ofstream solFile(ss.str());
            SaveToStream(sol, solFile);
            solFile.close();
            */
        }
timestat.timeOther.second += omp_get_wtime();

    }

```

```
timestat.timeOther.first += omp_get_wtime();
    wake.WakeSynchronize();

    for (int bou = 0; bou < boundary.size(); ++bou)
    {
        boundary[bou]->VirtualWakeSynchronize();
    }
timestat.timeOther.second += omp_get_wtime();



    //Вычисление скоростей вихрей: для тех, которые в следе, и виртуальных
    CalcVortexVelo(dt, timestat.timeCalcVortexConvVelo, timestat.timeCalcVortexDiffVelo);



    //Сохранение пелены до сдвига - для отладки
    size_t wakevirtualsize = wake.vtx.size();
    for (size_t bou = 0; bou < boundary.size(); bou++)
        wakevirtualsize += boundary[bou]->virtualWake.vtx.size();

timestat.timeOther.first += omp_get_wtime();

    std::vector<Vortex2D> vtx2;
    vtx2.reserve(wakevirtualsize);
    if (parallel.myidWork == 0)
    for (size_t i = 0; i < wake.vtx.size(); ++i)
    {
        vtx2.push_back(wake.vtx[i]);
    }
    if (parallel.myidWork == 0)
    for (size_t bou = 0; bou < boundary.size(); bou++)
    for (size_t i = 0; i < boundary[bou]->virtualWake.vtx.size(); ++i)
    {
        vtx2.push_back(boundary[bou]->virtualWake.vtx[i]);
    }
timestat.timeOther.second += omp_get_wtime();

    if (parallel.myidWork == 0)
    {
        if ((passport.timeDiscretizationProperties.saveTXT > 0) && (!(currentStep % passport.timeDiscretizationProperties.saveTXT)))
        {
            info('i') << "Saving kadr to txt-file " << std::endl;
            wake.SaveKadr(vtx2, passport.dir, currentStep, timestat.timeSaveKadr);
        }
        if ((passport.timeDiscretizationProperties.saveVTK > 0) && (!(currentStep % passport.timeDiscretizationProperties.saveVTK)))
        {
            info('i') << "Saving kadr to vtk-file " << std::endl;
            wake.SaveKadrVtk(vtx2, passport.dir, currentStep, timestat.timeSaveKadr);
        }
    }


/*  //Сохранятель слоев
    if (parallel.myidWork == 0)
    if (!(currentStep % passport.timeDiscretizationProperties.saveTXT))
    {
        std::string fname = fileNameStep("Bou", W.getPassport().timeDiscretizationProperties.nameLength, currentStep, "txt");

        std::ofstream outfile;
        outfile.open(passport.dir + "snapshots/" + fname);

        outfile << boundary.size() << '\n';
        for(size_t bou = 0; bou < boundary.size(); ++bou)
            outfile << airfoil[bou]->r.size() - 1 << ' ';
        outfile << '\n';

        for (size_t bou = 0; bou < boundary.size(); ++bou)
        for (size_t i = 0; i < airfoil[bou]->r.size() - 1; i++)
        {
            outfile << airfoil[bou]->r[i][0] << ' ' << airfoil[bou]->r[i][1] << ' ' << airfoil[bou]->r[i + 1][0] << ' ' << airfoil[bou]->r[i + 1][1] << ' ' << boundary[bou]->sheets.freeVortexSheet[i][0] << '\n';
        }//for i
        outfile.close();
    }
    */


    //Расчет и сохранение поля давления
```

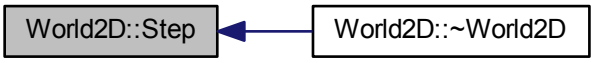
```
    if ((passport.timeDiscretizationProperties.saveVP > 0) && (!(currentStep % passport.timeDiscretizationProperties.saveVP)))
    {
        info('i') << "Saving VP to vtk-file " << std::endl;
        measureVP->CalcSaveVP(passport.dir, currentStep, timestat.timeVP);
    }

    //Вычисление сил, действующих на профиль и сохранение в файл
    if (parallel.myidWork == 0)
    for (size_t mech = 0; mech < mechanics.size(); ++mech)
    {
        mechanics[mech]->GetHydroDynamForce(timestat.timeGetHydroDynamForce);

timestat.timeOther.first += omp_get_wtime();
        mechanics[mech]->GenerateForcesString();
        mechanics[mech]->GeneratePositionString();
timestat.timeOther.second += omp_get_wtime();
    }

    std::vector<Point2D> newPos;

    //Движение вихрей (сброс вихрей + передвижение пелены)
    MoveVortexes(dt, newPos, timestat.timeMoveVortexes);


timestat.timeOther.first += omp_get_wtime();
    for (size_t afl = 0; afl < airfoil.size(); ++afl)
    {
        switch (passport.airfoilParams[afl].panelsType)
        {
        case 0:
            oldAirfoil.emplace_back(new AirfoilRect(*airfoil[afl]));
            break;
        case 1:

            //airfoil.emplace_back(new AirfoilCurv(GetPassport(), i, parallel));

            info('e') << "AirfoilCurv is not implemented now! " << std::endl;
            exit(1);
            break;
        }

        mechanics[afl]->Move();
    }//for

    for (size_t bou = 0; bou < boundary.size(); ++bou)
    {
        boundary[bou]->virtualWake.vtx.clear();
        boundary[bou]->virtualWake.vtx.resize(0);
    }
timestat.timeOther.second += omp_get_wtime();

    CheckInside(newPos, timestat.timeCheckInside);

timestat.timeOther.first += omp_get_wtime();
    //передача новых положений вихрей в пелену
    if (parallel.myidWork == 0)
    for (size_t i = 0; i < wake.vtx.size(); ++i)
        wake.vtx[i].r() = newPos[i];
timestat.timeOther.second += omp_get_wtime();

    //Сохранятель пелены ("старый")
//  if (parallel.myidWork == 0)
//  if (!(currentStep % passport.timeDiscretizationProperties.saveTXT))
//      wake.SaveKadr(wake.vtx, passport.dir, currentStep, timestat.timeSaveKadr);




// Определение параметров, отвечающих за увеличение радиуса коллапса

    std::vector<double> rightBorder, horizSpan;
    rightBorder.reserve(airfoil.size());
    horizSpan.reserve(airfoil.size());

    for (size_t q = 0; q < airfoil.size(); ++q)
    {
        rightBorder.push_back(airfoil[q]->upRight[0]);
        horizSpan.push_back(airfoil[q]->upRight[0] - airfoil[q]->lowLeft[0]);
    }

    wake.collapseRightBorderParameter = *std::max_element(rightBorder.begin(), rightBorder.end());
```

```
    wake.collapseScaleParameter = *std::max_element(horizSpan.begin(), horizSpan.end());

#if defined(__CUDACC__) || defined(USE_CUDA)
    cuda.setCollapseCoeff(wake.collapseRightBorderParameter, wake.collapseScaleParameter);
#endif

    wake.Restruct(timestat.timeRestruct);

    timestat.timeWakeSort.first = omp_get_wtime();

//Сортировка вихрей в пелене по абсциссе; было мнение, что позволяет оптимизировать счет на CUDA
/*
    std::sort(wake.vtx.begin(), wake.vtx.end(),
        [](const Vortex2D &a, const Vortex2D &b)
            {
                return a.r()[0] < b.r()[0];
            });
    */

    timestat.timeWakeSort.second = omp_get_wtime();

    timestat.timeOther.first += omp_get_wtime();
    oldAirfoil.clear();
    //oldAirfoil.resize(0);
    timestat.timeOther.second += omp_get_wtime();


    //Засечка времени в конце шага
    time.second = omp_get_wtime();


    if (parallel.myidWork == 0)
    {
        info('i') << "Step = " << currentStep \
            << " PhysTime = " << passport.physicalProperties.getCurrTime() \
            << " StepTime = " << Times::dT(time) << std::endl;

        timestat.GenerateStatString();
    }

    passport.physicalProperties.addCurrTime(dt);
    currentStep++;

}//Step()
```

Граф вызовов:



Граф вызова функции:



10.38.4 Данные класса

10.38.4.1 airfoil

std::vector<std::unique_ptr<Airfoil> > World2D::airfoil [private]

Список умных казателей на обтекаемые профили

См. определение в файле World2D.h строка 90

10.38.4.2 boundary

std::vector<std::unique_ptr<Boundary> > World2D::boundary [private]

Список умных указателей на формирователи граничных условий на профилях

См. определение в файле World2D.h строка 96

10.38.4.3 cuda

Gpu World2D::cuda [private]

Объект, управляющий графическим ускорителем

См. определение в файле World2D.h строка 132

10.38.4.4 currentStep

size_t World2D::currentStep [private]

Текущий номер шага в решаемой задаче

См. определение в файле World2D.h строка 135

10.38.4.5 info

LogStream World2D::info [mutable], [private]

Поток для вывода логов и сообщений об ошибках

См. определение в файле World2D.h строка 87

10.38.4.6 invMatr

Eigen::MatrixXd World2D::invMatr [private]

Обратная матрица

См. определение в файле World2D.h строка 117

10.38.4.7 matr

Eigen::MatrixXd World2D::matr [private]

Матрица системы

См. определение в файле World2D.h строка 114

10.38.4.8 measureVP

std::unique_ptr<MeasureVP> World2D::measureVP [private]

Поле скоростей и давления (для сохранения в файл)

См. определение в файле World2D.h строка 111

10.38.4.9 mechanics

std::vector<std::unique_ptr<Mechanics> > World2D::mechanics [private]

Список умных указателей на типы механической системы для каждого профиля

См. определение в файле World2D.h строка 99

10.38.4.10 oldAirfoil

std::vector<std::unique_ptr<Airfoil> > World2D::oldAirfoil [private]

Список умных казателей на обтекаемые профили для сохранения старого положения

См. определение в файле World2D.h строка 93

10.38.4.11 parallel

const Parallel& World2D::parallel [private]

Константная ссылка на параметры исполнения задачи в параллельном MPI-режиме

См. определение в файле World2D.h строка 129

10.38.4.12 passport

const Passport& World2D::passport [private]

Константная ссылка на паспорт конкретного расчета

См. определение в файле World2D.h строка 126

10.38.4.13 rhs

Eigen::VectorXd World2D::rhs [private]

Правая часть системы

См. определение в файле World2D.h строка 120

10.38.4.14 sol

Eigen::VectorXd World2D::sol [private]

Решение системы

См. определение в файле World2D.h строка 123

10.38.4.15 source

WakeDataBase World2D::source [private]

Источники

См. определение в файле World2D.h строка 108

10.38.4.16 timestat

Times World2D::timestat [private]

Сведения о временах выполнения основных операций

См. определение в файле World2D.h строка 138

10.38.4.17 velocity

std::unique_ptr<Velocity> World2D::velocity [private]

Умный укзатель на объект, определяющий методику вычисления скоростей

См. определение в файле World2D.h строка 102

10.38.4.18 wake

Wake World2D::wake [private]

Вихревой след

См. определение в файле World2D.h строка 105

Объявления и описания членов классов находятся в файлах:

- World2D/World2D.h
- World2D/World2D.cpp

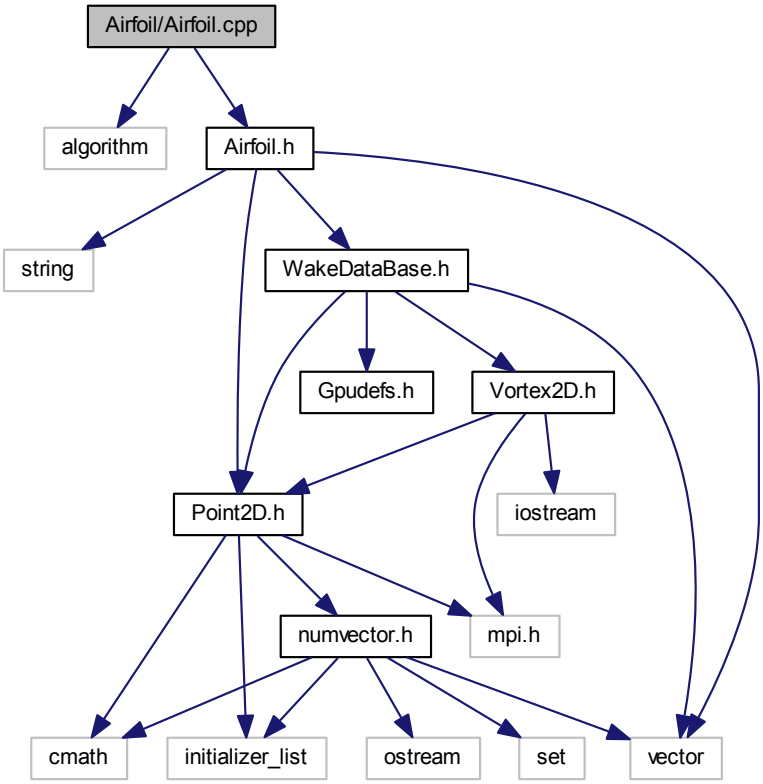
Глава 11

Файлы

11.1 Файл Airfoil/Airfoil.cpp

Файл кода с описанием класса Airfoil.

#include <algorithm>
#include "Airfoil.h"
Граф включаемых заголовочных файлов для Airfoil.cpp:



11.1.1 Подробное описание

Файл кода с описанием класса Airfoil.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

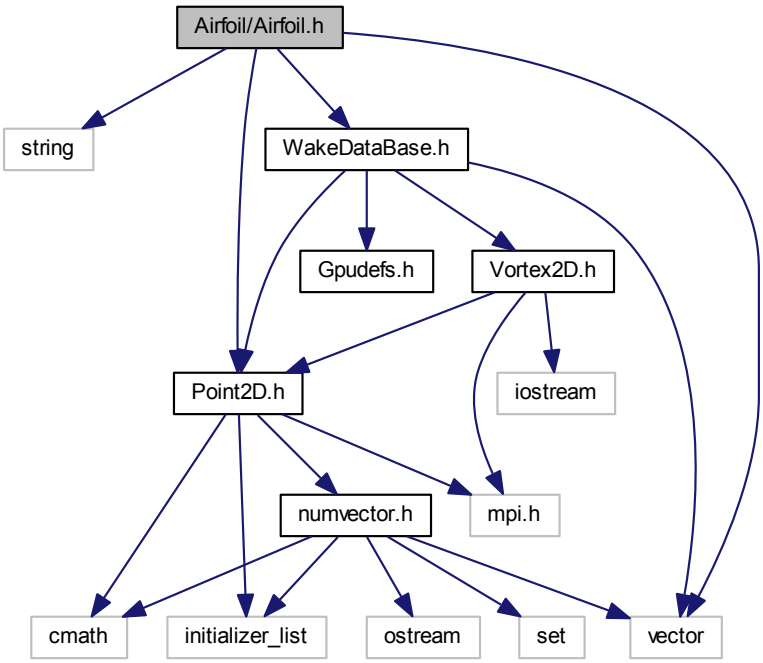
Дата

20 февраля 2019 г.

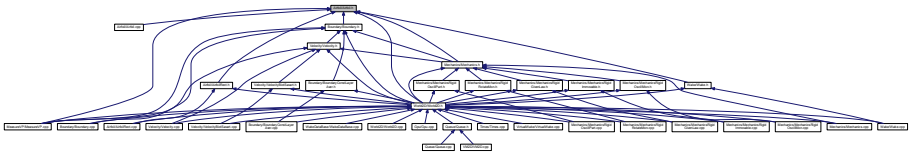
11.2 Файл Airfoil/Airfoil.h

Заголовочный файл с описанием класса Airfoil.

#include <string>
#include <vector>
#include "Point2D.h"
#include "WakeDataBase.h"
Граф включаемых заголовочных файлов для Airfoil.h:



Граф файлов, в которые включается этот файл:



Классы

- class Airfoil

  Абстрактный класс, определяющий обтекаемый профиль

11.2.1 Подробное описание

Заголовочный файл с описанием класса Airfoil.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

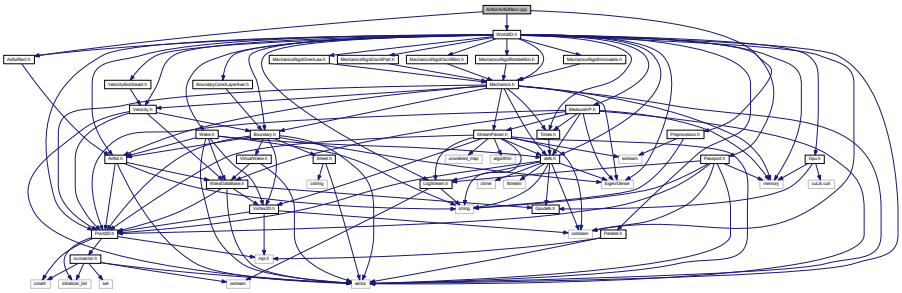
Дата

20 февраля 2019 г.

11.3 Файл Airfoil/AirfoilRect.cpp

Файл кода с описанием класса AirfoilRect.

#include "AirfoilRect.h"
#include "Preprocessor.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для AirfoilRect.cpp:



11.3.1 Подробное описание

Файл кода с описанием класса AirfoilRect.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

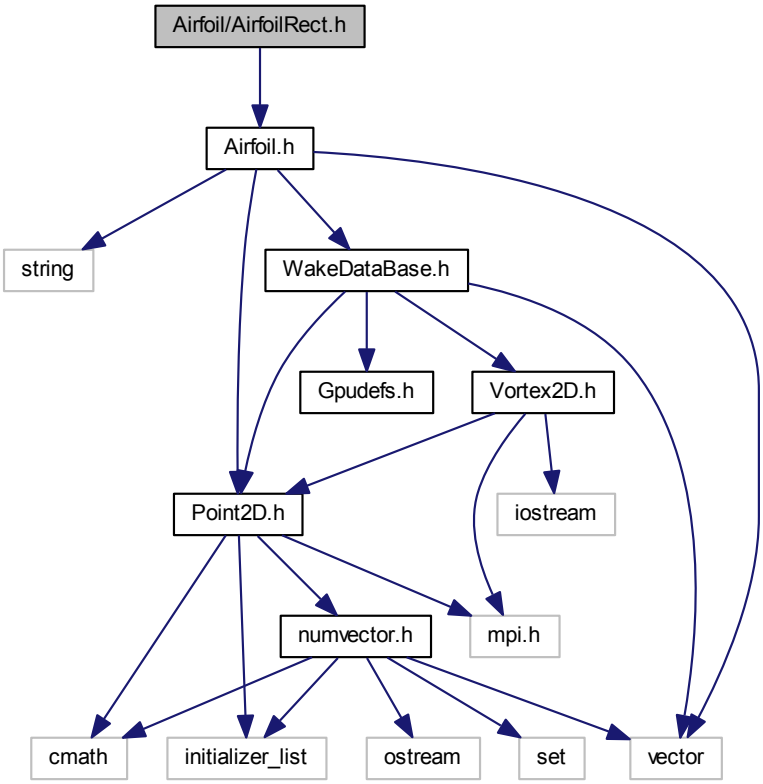
Дата

20 февраля 2019 г.

11.4 Файл Airfoil/AirfoilRect.h

Заголовочный файл с описанием класса AirfoilRect.

#include "Airfoil.h"
Граф включаемых заголовочных файлов для AirfoilRect.h:



Граф файлов, в которые включается этот файл:

Классы

- class AirfoilRect

  Класс, определяющий тип обтекаемого профиля

11.4.1 Подробное описание

Заголовочный файл с описанием класса AirfoilRect.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

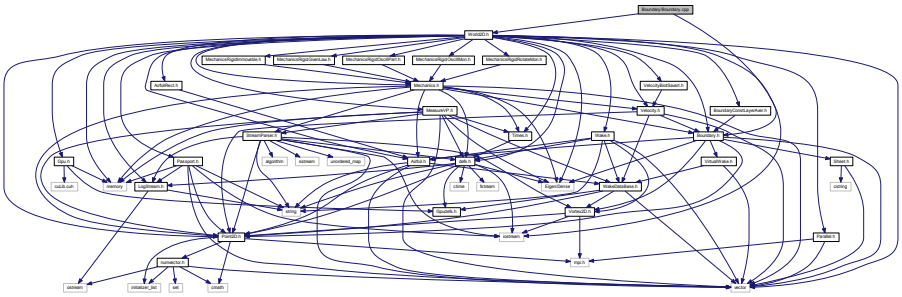
Дата

20 февраля 2019 г.

11.5 Файл Boundary/Boundary.cpp

Файл кода с описанием класса Boundary.

#include "Boundary.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для Boundary.cpp:



11.5.1 Подробное описание

Файл кода с описанием класса Boundary.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

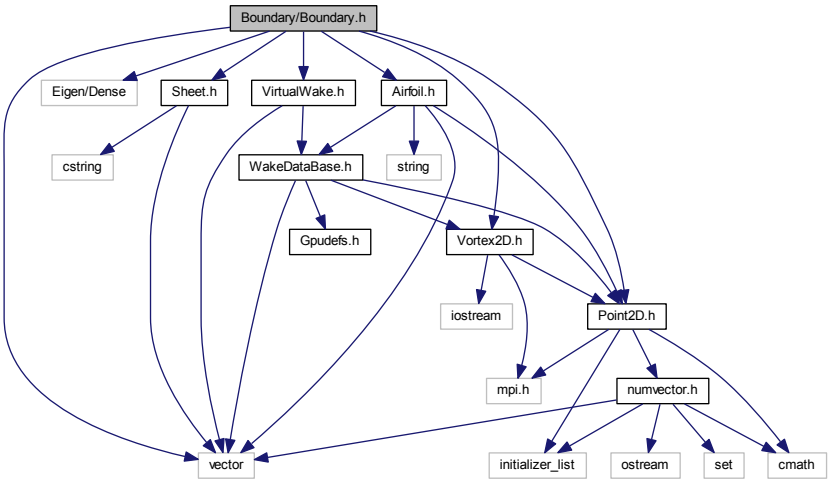
Дата

20 февраля 2019 г.

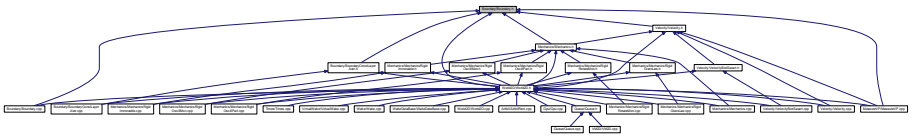
11.6 Файл Boundary/Boundary.h

Заголовочный файл с описанием класса Boundary.

#include <vector>
#include "Eigen/Dense"
#include "Airfoil.h"
#include "Point2D.h"
#include "Sheet.h"
#include "Vortex2D.h"
#include "VirtualWake.h"
Граф включаемых заголовочных файлов для Boundary.h:



Граф файлов, в которые включается этот файл:



Классы

- class Boundary

  Абстрактный класс, определяющий способ удовлетворения граничного условия на обтекаемом профиле

11.6.1 Подробное описание

Заголовочный файл с описанием класса Boundary.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

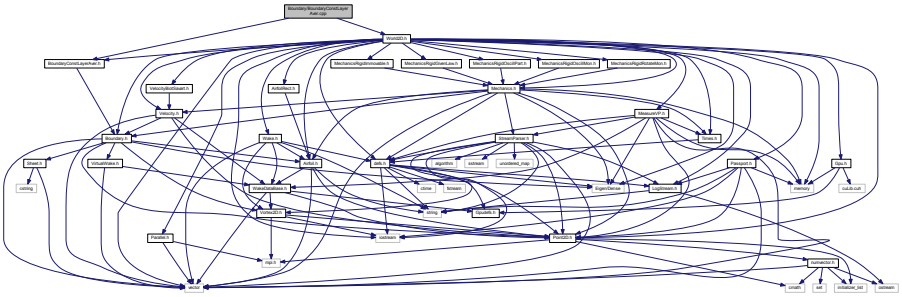
Дата

20 февраля 2019 г.

11.7 Файл Boundary/BoundaryConstLayerAver.cpp

Файл кода с описанием класса BoundaryConstLayerAver.

#include "BoundaryConstLayerAver.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для BoundaryConstLayerAver.cpp:



11.7.1 Подробное описание

Файл кода с описанием класса BoundaryConstLayerAver.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

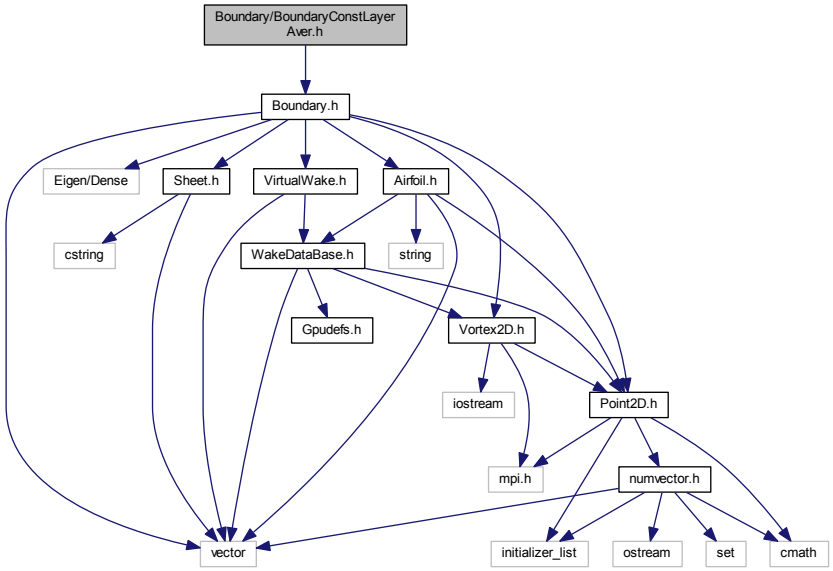
Дата

20 февраля 2019 г.

11.8 Файл Boundary/BoundaryConstLayerAver.h

Заголовочный файл с описанием класса BoundaryConstLayerAver.

#include "Boundary.h"
Граф включаемых заголовочных файлов для BoundaryConstLayerAver.h:



Граф файлов, в которые включается этот файл:

Классы

- class BoundaryConstLayerAver

  Класс, определяющий способ удовлетворения граничного условия на обтекаемом профиле

11.8.1 Подробное описание

Заголовочный файл с описанием класса BoundaryConstLayerAver.

Предупреждения

Пока реализованы не все методы

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

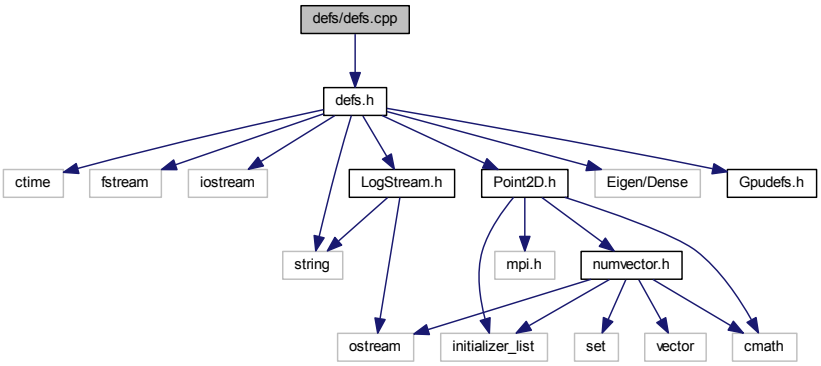
Дата

20 февраля 2019 г.

11.9 Файл defs/defs.cpp

Описание базовых вспомогательных функций

#include "defs.h"
Граф включаемых заголовочных файлов для defs.cpp:



Функции

- std::string CurrentDataTime ()

  Формирование строки с текущем временем и датой
- void PrintLogoToTextFile (std::ofstream &str, const std::string &fileName, const std::string &descr)

  Формирование заголовка файла программы VM2D.
- void PrintHeaderToTextFile (std::ofstream &str, const std::string &header)

  Формирование подзаголовка в текстовом файле вывода программы VM2D.
- void SaveToStream (const Eigen::MatrixXd &matr, std::ostream &str)

  Сохранение матрицы в поток
- void SaveToStream (const Eigen::MatrixXcd &matr, std::ostream &str)

  Сохранение комплекснозначной матрицы в поток
- void SaveToStream (const Eigen::VectorXd &vec, std::ostream &str)

  Сохранение вектора в поток
- void SaveToStream (const std::vector< Point2D > &vec, std::ostream &str)

  Сохранение списка из двумерных векторов (точек) в поток
- double M4 (double t)

  Ядро сглаживания (Монагана)

Переменные

- std::string Months [12] = { "January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December" }

11.9.1 Подробное описание

Описание базовых вспомогательных функций

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

11.9.2 Функции

11.9.2.1 CurrentDataTime()

std::string CurrentDataTime ( )

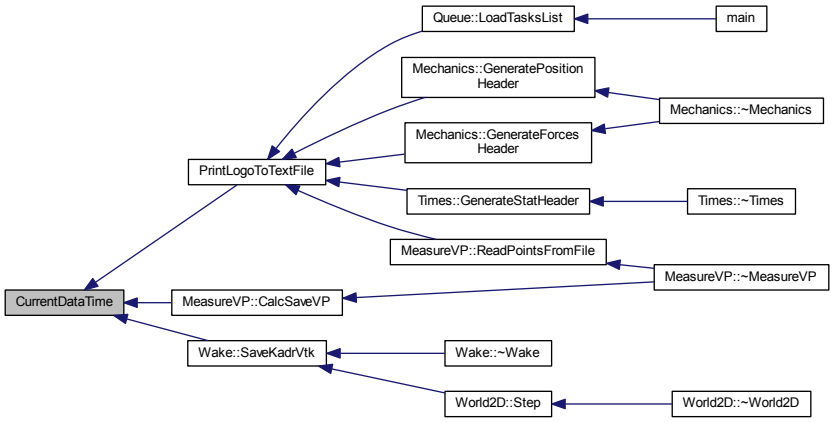
Формирование строки с текущем временем и датой

Возвращает строку с текущей датой и временем в формате XX Month 20XX at XX:XX:XX

См. определение в файле defs.cpp строка 45

```
46 {
47     std::time_t t = std::time(nullptr);
48
49 #pragma warning(push)
50 #pragma warning(disable : 4996) //_CRT_SECURE_NO_WARNINGS
51     std::tm tm = *std::localtime(&t);
52 #pragma warning(pop)
53
54     std::stringstream dateTimeStringStream;
55     //dateTimeStringStream << "| This file was created automatically " << std::put_time(&tm, "%d %B %Y") <<
       " at " << std::put_time(&tm, "%H:%M:%S");
56
57     std::stringstream st_day, st_hour, st_min, st_sec;
58     if (tm.tm_mday < 10)
59         st_day << "0" << tm.tm_mday;
60     else
61         st_day << tm.tm_mday;
62
63     if (tm.tm_hour < 10)
64         st_hour << "0" << tm.tm_hour;
65     else
66         st_hour << tm.tm_hour;
67
68     if (tm.tm_min < 10)
69         st_min << "0" << tm.tm_min;
70     else
71         st_min << tm.tm_min;
72
73     if (tm.tm_sec < 10)
74         st_sec << "0" << tm.tm_sec;
75     else
76         st_sec << tm.tm_sec;
77
78     dateTimeStringStream \
79         << st_day.str() << " " << Months[tm.tm_mon] << " " << 1900 + tm.tm_year \
80         << " at " \
81         << st_hour.str() << ":" << st_min.str() << ":" << st_sec.str();
82
83     //std::string dateTimeString;
84     //dateTimeStringStream >> dateTimeString;
85
86     return dateTimeStringStream.str();
87 }
```

Граф вызова функции:



11.9.2.2 M4()

double M4 (
double t)

Ядро сглаживания (Монагана)

Аргументы

in	t	аргумент ядра

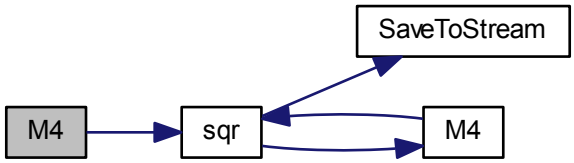
Возвращает

Значение ядра сглаживания в конкретной точке

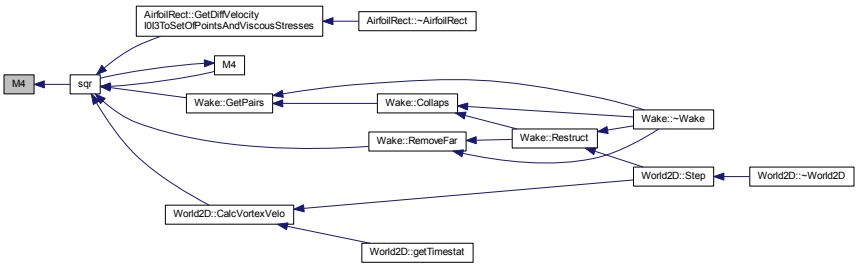
См. определение в файле defs.cpp строка 184

```
{
    double t2 = t*t;
    double mt = t > 0 ? t : -t;

    return (mt > 2.0) ? 0.0 : \
        (mt > 1.0) ? 0.5*sqr(2.0 - mt)*(1.0 - mt) : 1.0 - 2.5*t2 + 1.5*t2*mt;
}
```

Граф вызовов:



Граф вызова функции:



11.9.2.3 PrintHeaderToTextFile()

```
void PrintHeaderToTextFile (
            std::ofstream & str,
            const std::string & header )
```

Формирование подзаголовка в текстовом файле вывода программы VM2D.

Печатает в файл вывода программы VM2D подзаголовок и подчеркивает его

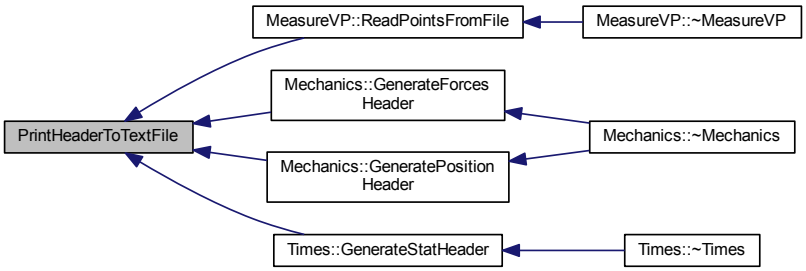
Аргументы

out	str	ссылка на файл для вывода, должен быть открыт
in	header	константная ссылка на строку с заголовком

См. определение в файле defs.cpp строка 134

```
135 {
136     str << '\n';
137     str << "// " << header << '\n';
138     str << "//";
139     for (size_t q = 0; q < header.length()+1; ++q)
140         str << '-';
141 }
```

Граф вызова функции:



11.9.2.4 PrintLogoToTextFile()

```
void PrintLogoToTextFile (
            std::ofstream & str,
            const std::string & fileName,
            const std::string & descr )
```

Формирование заголовка файла программы VM2D.

Печатает в шапку файла заголовок программы VM2D

Аргументы

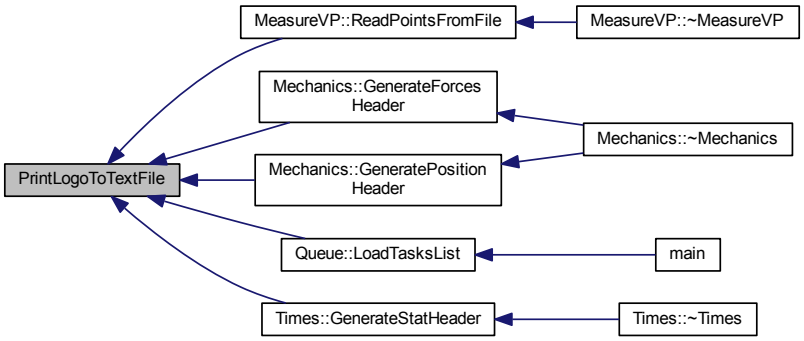
out	str	ссылка на файл для вывода, должен быть открыт
in	fileName	константная ссылка на строку с именем файла
in	descr	константная ссылка на строку с описанием файла

См. определение в файле defs.cpp строка 91

```
{
  str <<
     "/*-------------------------------*- VM2D -*-----------------*---------------*\\" << '\n' << \
     "| ##  ## ##   ##  ####  #####   |                          | Version 1.5    |" << '\n' << \
     "| ##  ## ### ### ##  ## ##  ##  |  VM2D: Vortex Method       | 2019/02/20     |" << '\n' << \
     "| ##  ## ## # ##   ##  ##  ##  |  for 2D Flow Simulation    *----------------*" << '\n' << \
     "|  ####  ##   ##   ##   ##  ##  |  Open Source Code                           |" << '\n' << \
     "|   ##   ##   ## ###### #####   |  https://www.github.com/vortexmethods/VM2D  |" << '\n' << \
      "|                                                                           |" << '\n' << \
      "| Copyright (C) 2017-2019 Ilia Marchevsky, Kseniia Kuzmina, Evgeniya Ryatina  |" << '\n' << \
      "*---------------------------------------------------------------------------*" << '\n';

  str <<
     "| File name: " << fileName;
  for (size_t q = 0; q < 65 - fileName.length(); ++q)
     str << " ";
  str <<
     "|"  << '\n';

  str <<
     "| Info: " << descr;
  for (size_t q = 0; q < 70 - descr.length(); ++q)
     str << " ";
  str <<
     "|" << '\n';


  std::string dateTimeString = "| This file was created automatically " + CurrentDataTime();
  //dateTimeStringStream >> dateTimeString;

  str << dateTimeString;
  for (size_t q = 0; q < 78 - dateTimeString.length(); ++q)
     str << " ";
  str <<
     "|" << '\n';

  str <<
     "\\*---------------------------------------------------------------------------*/" << '\n';
}//PrintLogoToTextFile(...)
```

Граф вызовов:

Граф вызова функции:



11.9.2.5 SaveToStream() [1/4]

```
void SaveToStream (
            const Eigen::MatrixXd & matr,
            std::ostream & str )
```

Сохранение матрицы в поток

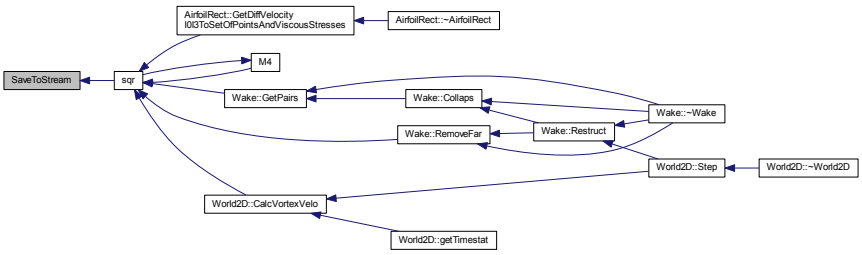
Аргументы

in	matr	константная ссылка на сохраняемую матрицу
in,out	str	ссылка на поток для сохранения

См. определение в файле defs.cpp строка 144

```
{
    for (int i = 0; i < matr.rows(); ++i)
    {
        for (int j = 0; j < matr.cols(); ++j)
            str << matr(i, j) << " ";
        str << std::endl;
    }
}//SaveToStream(...)
```

Граф вызова функции:



11.9.2.6 SaveToStream() [2/4]

```
void SaveToStream (
            const Eigen::MatrixXcd & matr,
            std::ostream & str )
```

Сохранение комплекснозначной матрицы в поток

Аргументы

in	matr	константная ссылка на сохраняемую комплекснозначную матрицу
in,out	str	ссылка на поток для сохранения

См. определение в файле defs.cpp строка 156

```
157 {
158     for (int i = 0; i < matr.rows(); ++i)
159     {
160         for (int j = 0; j < matr.cols(); ++j)
161             str << matr(i, j) << " ";
162         str << std::endl;
163     }
164 }//SaveToStream(...)
```

11.9.2.7 SaveToStream() [3/4]

```
void SaveToStream (
            const Eigen::VectorXd & vec,
            std::ostream & str )
```

Сохранение вектора в поток

Аргументы

in	vec	константная ссылка на сохраняемый вектор
in,out	str	ссылка на поток для сохранения

См. определение в файле defs.cpp строка 168

```
169 {
170     for (int i = 0; i < vec.size(); ++i)
171         str << vec(i) << " ";
172 }//SaveToStream(...)
```

11.9.2.8 SaveToStream() [4/4]

```
void SaveToStream (
            const std::vector< Point2D > & vec,
            std::ostream & str )
```

Сохранение списка из двумерных векторов (точек) в поток

Аргументы

in	vec	константная ссылка на сохраняемый список из двумерных векторов (точек)
in,out	str	ссылка на поток для сохранения

См. определение в файле defs.cpp строка 176

```
177 {
178     for (size_t i = 0; i < vec.size(); ++i)
179         str << "{ " << vec[i][0] << " " << vec[i][1] << " } ";
180 }//SaveToStream(...)
```

11.9.3 Переменные

11.9.3.1 Months

```
std::string Months[12] = { "January", "February", "March", "April", "May", "June", "July", "August", "September",
"October", "November", "December" }
```

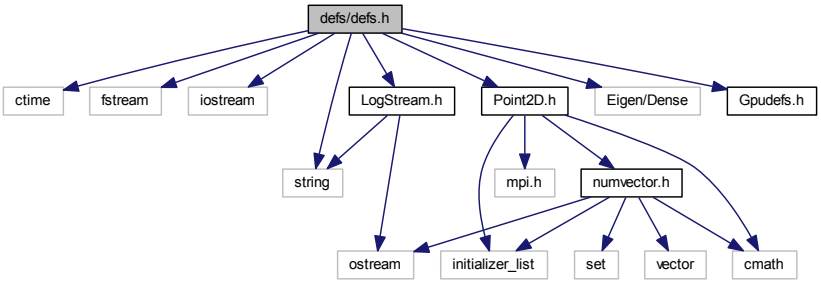
См. определение в файле defs.cpp строка 42

11.10 Файл defs/defs.h

Описание базовых вспомогательных функций

```
#include <ctime>
#include <fstream>
#include <iostream>
#include <string>
#include "Eigen/Dense"
#include "Gpudefs.h"
#include "Point2D.h"
```

#include "LogStream.h"
Граф включаемых заголовочных файлов для defs.h:



Граф файлов, в которые включается этот файл:

Пространства имен

- defaults

  Глобальные параметры по умолчанию

Макросы

- #define DYN_SCHEDULE 20

  параметр расписания для распараллеливания циклов по OpenMP.

Определения типов

- typedef std::pair< double, double > timePeriod

  Тип для хранения начала и конца промежутка времени

Функции

- const std::vector< std::string > defaults::defaultFileAirfoil ({})

  Список профилей
- const std::string defaults::defaultFileWake ("")

  Файл со следом
- const std::string defaults::defaultFileSource ("")

  Файл с источниками
- const std::vector< std::string > defaults::defaultAirfoil ({})

Список профилей

- std::string CurrentDataTime ()

  Формирование строки с текущем временем и датой
- void PrintLogoToTextFile (std::ofstream &str, const std::string &fileName, const std::string &descr)

  Формирование заголовка файла программы VM2D.
- void PrintHeaderToTextFile (std::ofstream &str, const std::string &header)

  Формирование подзаголовка в текстовом файле вывода программы VM2D.
- template<typename T >
  std::ostream & operator<< (std::ostream &_stream, const std::vector< T > &_vec)

  Переопределение оператора "<<" для вывода в поток вектора std::vector.
- bool fileExistTest (const std::string &fileName, LogStream &info)

  Проверка существования файла
- std::string fileNameStep (const std::string &name, int length, size_t step, const std::string &ext)

  Формирование имени файла
- template<typename T >
  T sqr (T x)

  Возведение числа в квадрат
- void SaveToStream (const Eigen::MatrixXd &matr, std::ostream &str)

  Сохранение матрицы в поток
- void SaveToStream (const Eigen::MatrixXcd &matr, std::ostream &str)

  Сохранение комплекснозначной матрицы в поток
- void SaveToStream (const Eigen::VectorXd &vec, std::ostream &str)

  Сохранение вектора в поток
- void SaveToStream (const std::vector< Point2D > &vec, std::ostream &str)

  Сохранение списка из двумерных векторов (точек) в поток
- double M4 (double t)

  Ядро сглаживания (Монагана)
- double boundDenom (double r2, double eps2)

  Способ сглаживания скорости вихря (вихрь Рэнкина или вихрь Ламба)

Переменные

- const double PI = 3.14159265358979323846264338327 95

  Число π.
- const double IDPI = 0.159154943091895335768883763372 5

  Число $\frac{1}{2\pi}$.
- const double DPI = 6.283185307179586476925286766559 0

  Число 2π.
- const double defaults::defaultTimeStart = 0.0

  Начало расчета
- const double defaults::defaultTimeAccel = 0.0

  Время разгона
- const int defaults::defaultSaveTXT = 1

  Шаг сохранения в текстовый файл
- const int defaults::defaultSaveVTK = 0

  Шаг сохранения в бинарный файл
- const int defaults::defaultSaveVP = 0

  Шаг подсчета поля скорости и давления
- const int defaults::defaultNameLength = 5

  Число разрядов в имени файла

- const double defaults::defaultDistFar = 10.0

  Радиус убивания дальнего следа
- const double defaults::defaultDelta = 1.e-5

  Расстояние, на которое рождаемый вихрь отодвигается от профиля
- const int defaults::defaultVortexPerPanel = 1

  Число вихрей, рождаемых на одной панели
- const double defaults::defaultVRef = 0.0

  Референсная скорость, равная нулю, что означает ее равенство скорости набегающего потока
- const std::string defaults::defaultAirfoilsDir = "./airfoils/"

  Каталог с файлами профилей
- const std::string defaults::defaultWakesDir = "./wakes/"

  Каталог с файлами вихревых следов
- const Point2D defaults::defaultBasePoint = { 0.0, 0.0 }

  Базовое смещение профиля
- const double defaults::defaultScale = 1.0

  Коэффициент масштабирования профиля
- const double defaults::defaultAngle = 0.0

  Угол атаки
- const int defaults::defaultPanelsType = 0

  Тип панелей
- const bool defaults::defaultInverse = false

  Признак разворота нормалей (для расчета внутреннего течения)
- const int defaults::defaultBoundaryCondition = 0

  Способ удовлетворения граничного условия
- const int defaults::defaultMechanicalSystemType = 0

  Тип механической системы
- const std::string defaults::defaultMechanicalSystem = ""
- const std::string defaults::defaultPspFile = "passport"

  Имя файла с паспортом задачи
- const int defaults::defaultNp = 1

  Необходимое число процессоров для решения задачи
- const std::string defaults::defaultCopyPspFile = ""

  Имя файла с паспортом задачи для копирования в новые каталоги
- static std::ostream * defaults::defaultQueueLogStream = &std::cout

  Поток вывода логов и ошибок очереди
- static std::ostream * defaults::defaultWorld2DLogStream = &std::cout

  Поток вывода логов и ошибок задачи

11.10.1 Подробное описание

Описание базовых вспомогательных функций

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

11.10.2 Макросы

11.10.2.1 DYN_SCHEDULE

#define DYN_SCHEDULE 20

параметр расписания для распараллеливания циклов по OpenMP.

См. определение в файле defs.h строка 55

11.10.3 Типы

11.10.3.1 timePeriod

typedef std::pair<double, double> timePeriod

Тип для хранения начала и конца промежутка времени

См. определение в файле defs.h строка 58

11.10.4 Функции

11.10.4.1 boundDenom()

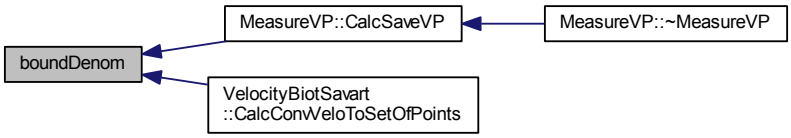
```
double boundDenom (
            double r2,
            double eps2 )   [inline]
```

Способ сглаживания скорости вихря (вихрь Рэнкина или вихрь Ламба)

См. определение в файле defs.h строка 302

```
{
#ifndef LAMBVORTEX
    return fmax(r2, eps2);
#else
    if (r2 > eps2)
        return fmax(r2, eps2);
    else
        return (r2 < 1e-10) ? 1e-10 : r2 / (1.0 - exp(-6.0*r2 / eps2));
#endif
}
```

Граф вызова функции:



11.10.4.2 CurrentDataTime()

std::string CurrentDataTime ( )

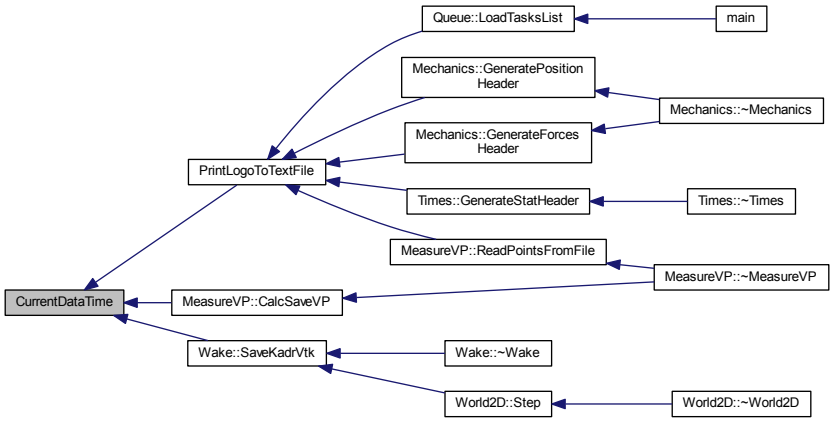
Формирование строки с текущем временем и датой

Возвращает строку с текущей датой и временем в формате XX Month 20XX at XX:XX:XX

См. определение в файле defs.cpp строка 45

```
46 {
47     std::time_t t = std::time(nullptr);
48
49 #pragma warning(push)
50 #pragma warning(disable : 4996) //_CRT_SECURE_NO_WARNINGS
51     std::tm tm = *std::localtime(&t);
52 #pragma warning(pop)
53
54     std::stringstream dateTimeStringStream;
55     //dateTimeStringStream << "| This file was created automatically " << std::put_time(&tm, "%d %B %Y") <<
       " at " << std::put_time(&tm, "%H:%M:%S");
56
57     std::stringstream st_day, st_hour, st_min, st_sec;
58     if (tm.tm_mday < 10)
59         st_day << "0" << tm.tm_mday;
60     else
61         st_day << tm.tm_mday;
62
63     if (tm.tm_hour < 10)
64         st_hour << "0" << tm.tm_hour;
65     else
66         st_hour << tm.tm_hour;
67
68     if (tm.tm_min < 10)
69         st_min << "0" << tm.tm_min;
70     else
71         st_min << tm.tm_min;
72
73     if (tm.tm_sec < 10)
74         st_sec << "0" << tm.tm_sec;
75     else
76         st_sec << tm.tm_sec;
77
78     dateTimeStringStream \
79         << st_day.str() << " " << Months[tm.tm_mon] << " " << 1900 + tm.tm_year \
80         << " at " \
81         << st_hour.str() << ":" << st_min.str() << ":" << st_sec.str();
82
83     //std::string dateTimeString;
84     //dateTimeStringStream >> dateTimeString;
85
86     return dateTimeStringStream.str();
87 }
```

Граф вызова функции:



11.10.4.3 fileExistTest()

```
bool fileExistTest (
            const std::string & fileName,
            LogStream & info )  [inline]
```

Проверка существования файла

Аргументы

in	fileName	константная ссылка на имя проверяемого файла
in,out	info	ссылка на поток вывода логов/ошибок

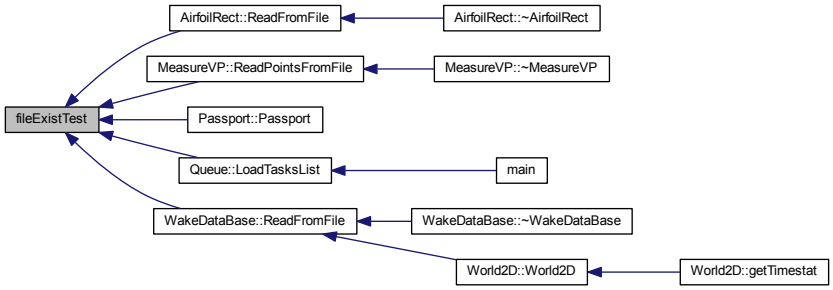
Возвращает

true, если файл существует; false если файл отсутствует

См. определение в файле defs.h строка 218

```
219 {
220     std::ifstream f(fileName.c_str());
221     if (f.good())
222     {
223         f.close();
224         f.clear();
225
226         info('i') << "file " << fileName << " is found" << std::endl;
227         return true;
228     }
229     else
230     {
231         info('e') << "file " << fileName << " is not found" << std::endl;
232         exit(-1);
233         return false;
234     }
235 }
```

Граф вызова функции:



11.10.4.4 fileNameStep()

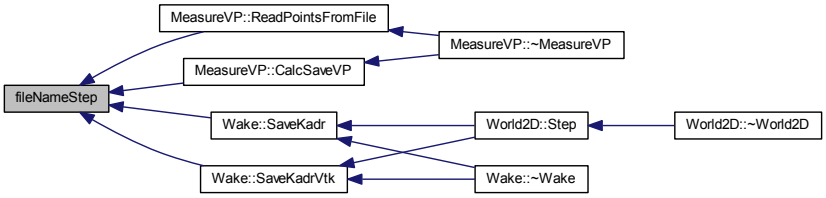
```
std::string fileNameStep (
            const std::string & name,
            int length,
            size_t step,
            const std::string & ext )   [inline]
```

Формирование имени файла

См. определение в файле defs.h строка 241

```
{
    std::string fname(name);

    size_t dec = 1;

    for (int i = 1; i < length; ++i)
    {
        dec *= 10;
        if (step < dec)
            fname += "0";
    }

    std::ostringstream ss;
    ss << step;
    fname += ss.str();
    fname += ".";
    fname += ext;

    return fname;
}
```

Граф вызова функции:



11.10.4.5 M4()

```
double M4 (
            double t )
```

Ядро сглаживания (Монагана)

Аргументы

in	t	аргумент ядра

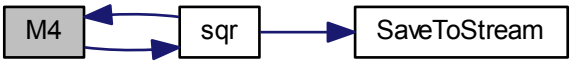
Возвращает

Значение ядра сглаживания в конкретной точке

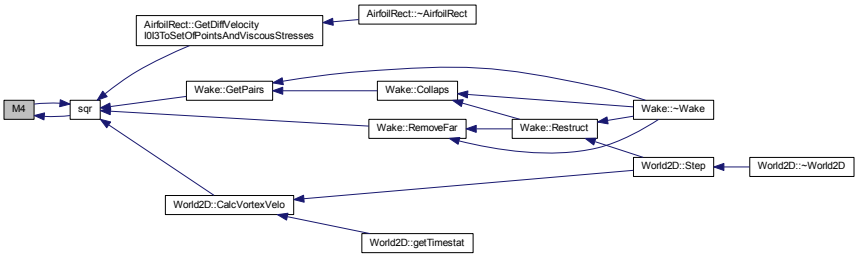
См. определение в файле defs.cpp строка 184

```
{
    double t2 = t*t;
    double mt = t > 0 ? t : -t;

    return (mt > 2.0) ? 0.0 : \
        (mt > 1.0) ? 0.5*sqr(2.0 - mt)*(1.0 - mt) : 1.0 - 2.5*t2 + 1.5*t2*mt;
}
```

Граф вызовов:



Граф вызова функции:



11.10.4.6 operator<<()

```
template<typename T >
std::ostream& operator<< (
            std::ostream & _stream,
            const std::vector< T > & _vec )
```

Переопределение оператора "<<" для вывода в поток вектора std::vector.

Компоненты вектора выводятся в фигурных скобках с разделителем ";"

Template Parameters

T	тип элементов в векторе

Аргументы

in,out	_stream	ссылка на поток для вывода
in	_vec	константная ссылка на выводимый вектор

Возвращает

ссылка на поток

См. определение в файле defs.h строка 197

```
198 {
199     size_t n = _vec.size();
200     _stream << "{ ";
201     if (n > 0)
202     {
203         for (size_t j = 0; j < n - 1; ++j)
204             _stream << _vec[j] << ", ";
205         _stream << _vec[n - 1];
206     }
207     _stream << " }";
208     return _stream;
209 }//operator<<(...)
```

11.10.4.7 PrintHeaderToTextFile()

```
void PrintHeaderToTextFile (
                std::ofstream & str,
                const std::string & header )
```

Формирование подзаголовка в текстовом файле вывода программы VM2D.

Печатает в файл вывода программы VM2D подзаголовок и подчеркивает его

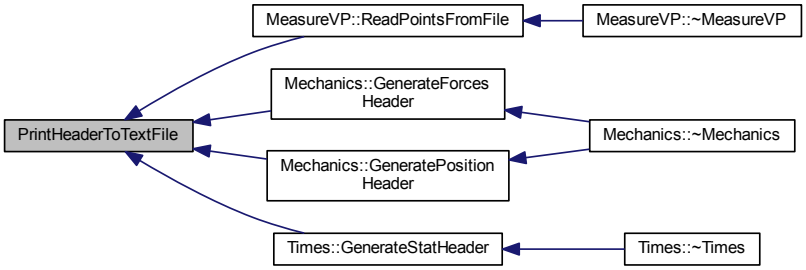
Аргументы

out	str	ссылка на файл для вывода, должен быть открыт
in	header	константная ссылка на строку с заголовком

См. определение в файле defs.cpp строка 134

```
135 {
136     str << '\n';
137     str << "// " << header << '\n';
138     str << "//";
139     for (size_t q = 0; q < header.length()+1; ++q)
140         str << '-';
141 }
```

Граф вызова функции:



11.10.4.8 PrintLogoToTextFile()

```
void PrintLogoToTextFile (
            std::ofstream & str,
            const std::string & fileName,
            const std::string & descr )
```

Формирование заголовка файла программы VM2D.

Печатает в шапку файла заголовок программы VM2D

Аргументы

out	str	ссылка на файл для вывода, должен быть открыт
in	fileName	константная ссылка на строку с именем файла
in	descr	константная ссылка на строку с описанием файла

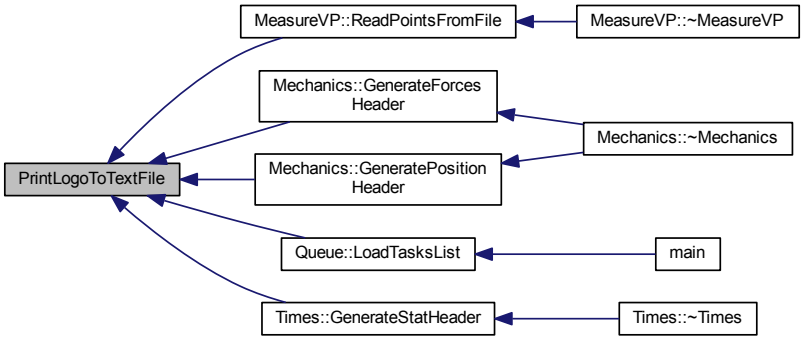
См. определение в файле defs.cpp строка 91

```
92 {
93     str <<
94         "/*--------------------------------*- VM2D -*-----------------*----------------*\\" << '\n' << \
95         "| ##  ## ##   ##  ####  #####   |                          | Version 1.5    |" << '\n' << \
96         "| ##  ## ### ### ##  ## ##  ##  |  VM2D: Vortex Method      | 2019/02/20     |" << '\n' << \
97         "| ##  ## ## # ##    ##  ##  ##  |  for 2D Flow Simulation    *----------------*" << '\n' << \
98         "|  ####  ##   ##   ##   ##  ##  |  Open Source Code                            |" << '\n' << \
99         "|   ##   ##   ## ###### #####   |  https://www.github.com/vortexmethods/VM2D  |" << '\n' << \
100         "|                                                                             |" << '\n' << \
101         "| Copyright (C) 2017-2019 Ilia Marchevsky, Kseniia Kuzmina, Evgeniya Ryatina  |" << '\n' << \
102         "*-----------------------------------------------------------------------------*" << '\n';
103
104     str <<
105         "| File name: " << fileName;
106     for (size_t q = 0; q < 65 - fileName.length(); ++q)
107         str << " ";
108     str <<
109         "|"  << '\n';
110
111     str <<
112         "| Info: " << descr;
113     for (size_t q = 0; q < 70 - descr.length(); ++q)
114         str << " ";
```

```
    str <<
        "|" << '\n';


    std::string dateTimeString = "| This file was created automatically " +
CurrentDataTime();
    //dateTimeStringStream >> dateTimeString;

    str << dateTimeString;
    for (size_t q = 0; q < 78 - dateTimeString.length(); ++q)
        str << " ";
    str <<
        "|" << '\n';

    str <<
        "\\*-----------------------------------------------------------------------*/" << '\n';
}//PrintLogoToTextFile(...)
```

Граф вызовов:

Граф вызова функции:



11.10.4.9 SaveToStream() [1/4]

void SaveToStream (
const Eigen::MatrixXd & matr,
std::ostream & str)

Сохранение матрицы в поток

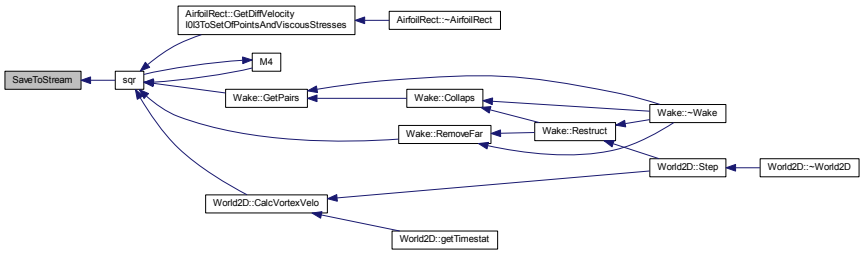
Аргументы

in	matr	константная ссылка на сохраняемую матрицу
in,out	str	ссылка на поток для сохранения

См. определение в файле defs.cpp строка 144

```
145 {
146     for (int i = 0; i < matr.rows(); ++i)
147     {
148         for (int j = 0; j < matr.cols(); ++j)
149             str << matr(i, j) << " ";
150         str << std::endl;
151     }
152 }//SaveToStream(...)
```

Граф вызова функции:



11.10.4.10 SaveToStream() [2/4]

void SaveToStream (
const Eigen::MatrixXcd & matr,
std::ostream & str)

Сохранение комплекснозначной матрицы в поток

Аргументы

in	matr	константная ссылка на сохраняемую комплекснозначную матрицу
in,out	str	ссылка на поток для сохранения

См. определение в файле defs.cpp строка 156

```
157 {
158     for (int i = 0; i < matr.rows(); ++i)
159     {
160         for (int j = 0; j < matr.cols(); ++j)
161             str << matr(i, j) << " ";
162         str << std::endl;
163     }
164 }//SaveToStream(...)
```

11.10.4.11 SaveToStream() [3/4]

```
void SaveToStream (
            const Eigen::VectorXd & vec,
            std::ostream & str )
```

Сохранение вектора в поток

Аргументы

in	vec	константная ссылка на сохраняемый вектор
in,out	str	ссылка на поток для сохранения

См. определение в файле defs.cpp строка 168

```
{
    for (int i = 0; i < vec.size(); ++i)
        str << vec(i) << " ";
}//SaveToStream(...)
```

11.10.4.12 SaveToStream() [4/4]

```
void SaveToStream (
            const std::vector< Point2D > & vec,
            std::ostream & str )
```

Сохранение списка из двумерных векторов (точек) в поток

Аргументы

in	vec	константная ссылка на сохраняемый список из двумерных векторов (точек)
in,out	str	ссылка на поток для сохранения

См. определение в файле defs.cpp строка 176

```
{
    for (size_t i = 0; i < vec.size(); ++i)
        str << "{ " << vec[i][0] << " " << vec[i][1] << " } ";
}//SaveToStream(...)
```

11.10.4.13 sqr()

```
template<typename T >
T sqr (
            T x )   [inline]
```

Возведение числа в квадрат

Template Parameters

T	тип данных

Аргументы

in	x	аргумент

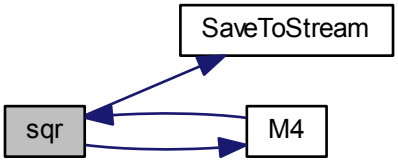
Возвращает

квадрат аргумента x^2

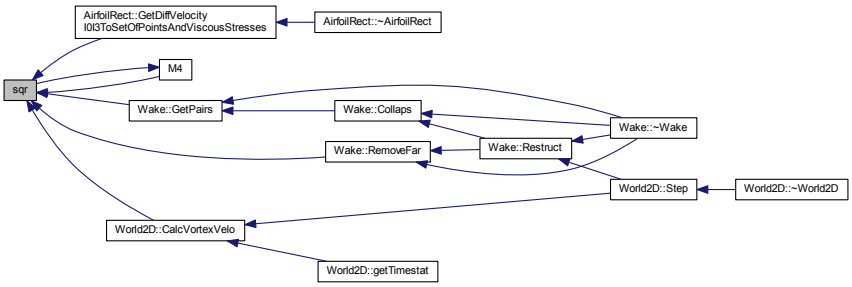
См. определение в файле defs.h строка 269

```
269 { return x*x; }
```

Граф вызовов:



Граф вызова функции:



11.10.5 Переменные

11.10.5.1 DPI

const double DPI = 6.28318530717958647692528676655 90

Число 2π.

См. определение в файле defs.h строка 67

11.10.5.2 IDPI

const double IDPI = 0.15915494309189533576888376337 25

Число $\frac{1}{2\pi}$.

См. определение в файле defs.h строка 64

11.10.5.3 PI

const double PI = 3.14159265358979323846264338327 95

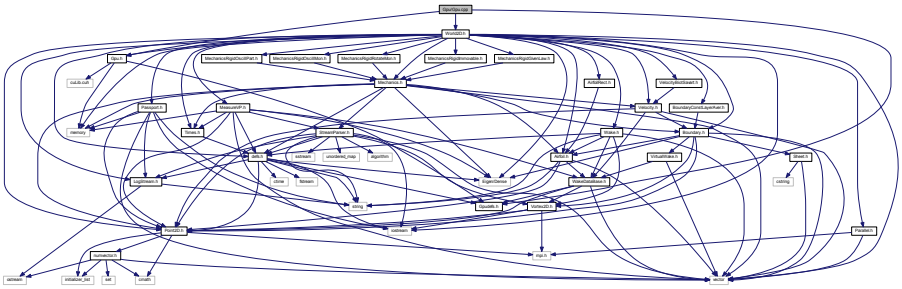
Число π.

См. определение в файле defs.h строка 61

11.11 Файл Gpu/Gpu.cpp

Файл кода с описанием класса Gpu.

#include "Gpu.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для Gpu.cpp:



11.11.1 Подробное описание

Файл кода с описанием класса Gpu.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

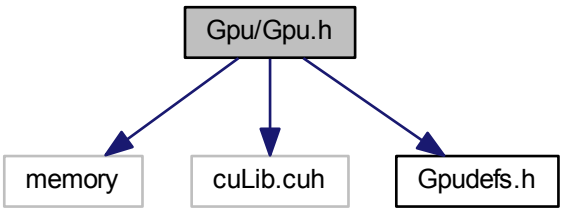
Дата

20 февраля 2019 г.

11.12 Файл Gpu/Gpu.h

Заголовочный файл с описанием класса Gpu.

#include <memory>
#include "cuLib.cuh"
#include "Gpudefs.h"
Граф включаемых заголовочных файлов для Gpu.h:



Граф файлов, в которые включается этот файл:

Классы

- class Gpu

  Класс, обеспечивающий возможность выполнения вычислений на GPU по технологии Nvidia C↩ UDA.

11.12.1 Подробное описание

Заголовочный файл с описанием класса Gpu.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

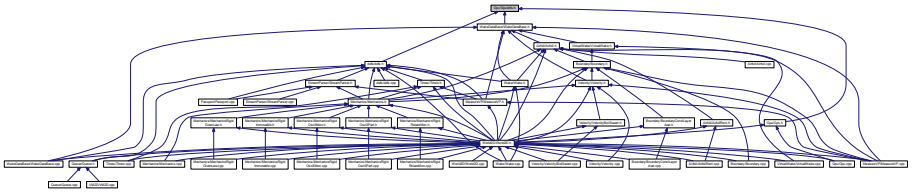
Дата

20 февраля 2019 г.

11.13 Файл Gpu/Gpudefs.h

Описание констант и параметров для взаимодействия с графическим ускорителем

Граф файлов, в которые включается этот файл:



Макросы

- #define CUDA_ENABLE(a) a

  вихрь Ламба
- #define CUDA_DISABLE(a)
- #define CUDAdef CUDA_DISABLE
- #define IFCUDA_(e, a) e(a)
- #define IFCUDA(...) IFCUDA_(CUDAdef, __VA_ARGS__)
- #define CUBLOCK (128)
- #define INC_VORT_DEV (1024)
- #define CU_I1I2
- #define CU_RHS
- #define CU_I0I3
- #define CU_CONV_TOWAKE
- #define CU_CONV_TOBOU
- #define CU_CONVVIRT
- #define CU_PAIRS

11.13.1 Подробное описание

Описание констант и параметров для взаимодействия с графическим ускорителем

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

11.13.2 Макросы

11.13.2.1 CU_CONV_TOBOU

#define CU_CONV_TOBOU

См. определение в файле Gpudefs.h строка 75

11.13.2.2 CU_CONV_TOWAKE

#define CU_CONV_TOWAKE

См. определение в файле Gpudefs.h строка 74

11.13.2.3 CU_CONVVIRT

#define CU_CONVVIRT

См. определение в файле Gpudefs.h строка 76

11.13.2.4 CU_I0I3

#define CU_I0I3

См. определение в файле Gpudefs.h строка 73

11.13.2.5 CU_I1I2

#define CU_I1I2

См. определение в файле Gpudefs.h строка 71

11.13.2.6 CU_PAIRS

#define CU_PAIRS

См. определение в файле Gpudefs.h строка 77

11.13.2.7 CU_RHS

#define CU_RHS

См. определение в файле Gpudefs.h строка 72

11.13.2.8 CUBLOCK

#define CUBLOCK (128)

См. определение в файле Gpudefs.h строка 68

11.13.2.9 CUDA_DISABLE

#define CUDA_DISABLE(
a)

См. определение в файле Gpudefs.h строка 56

11.13.2.10 CUDA_ENABLE

```
#define CUDA_ENABLE(
            a ) a
```

вихрь Ламба

признак использования CUDA

Устанавливается автоматически средствами cmake Если на компьютере найден CUDA Toolkit, то программа будет использовать графическую карту. Чтобы этого не происходило, надо принудительно сделать раскомментировать undef USE_CUDA

См. определение в файле Gpudefs.h строка 55

11.13.2.11 CUDAdef

```
#define CUDAdef CUDA_DISABLE
```

См. определение в файле Gpudefs.h строка 61

11.13.2.12 IFCUDA

```
#define IFCUDA(
            ... ) IFCUDA_(CUDAdef, __VA_ARGS__)
```

См. определение в файле Gpudefs.h строка 65

11.13.2.13 IFCUDA_

```
#define IFCUDA_(
            e,
            a ) e(a)
```

См. определение в файле Gpudefs.h строка 64

11.13.2.14 INC_VORT_DEV

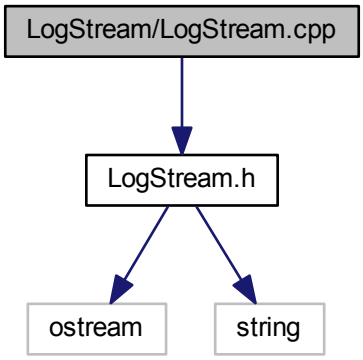
```
#define INC_VORT_DEV (1024)
```

См. определение в файле Gpudefs.h строка 69

11.14 Файл LogStream/LogStream.cpp

Файл кода с описанием класса LogStream.

#include "LogStream.h"
Граф включаемых заголовочных файлов для LogStream.cpp:



11.14.1 Подробное описание

Файл кода с описанием класса LogStream.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

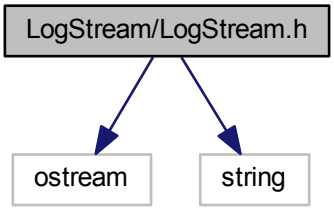
Дата

20 февраля 2019 г.

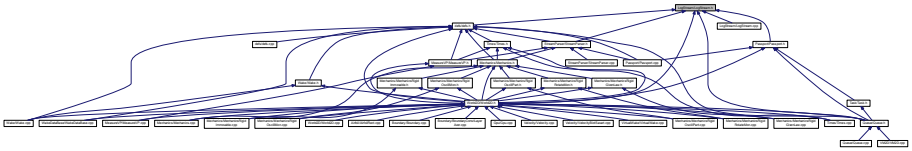
11.15 Файл LogStream/LogStream.h

Заголовочный файл с описанием класса LogStream.

#include <ostream>
#include <string>
Граф включаемых заголовочных файлов для LogStream.h:



Граф файлов, в которые включается этот файл:



Классы

- class LogStream

  Класс, определяющий работу с потоком логов

11.15.1 Подробное описание

Заголовочный файл с описанием класса LogStream.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

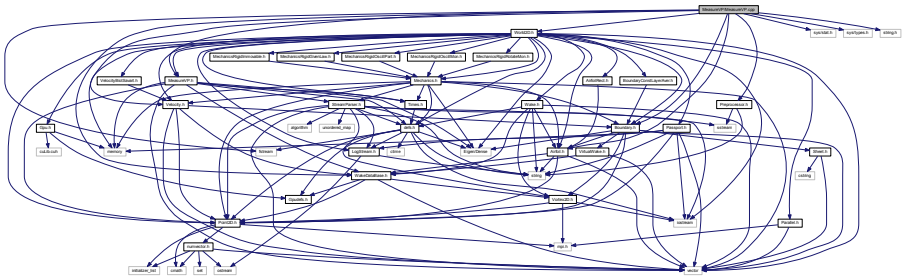
20 февраля 2019 г.

11.16 Файл MeasureVP/MeasureVP.cpp

Файл кода с описанием класса MeasureVP.

```
#include <fstream>
#include <sys/stat.h>
#include <sys/types.h>
#include "MeasureVP.h"
#include "World2D.h"
#include "Airfoil.h"
#include "Velocity.h"
#include "Boundary.h"
#include "Preprocessor.h"
#include "WakeDataBase.h"
#include <string.h>
```

Граф включаемых заголовочных файлов для MeasureVP.cpp:



11.16.1 Подробное описание

Файл кода с описанием класса MeasureVP.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

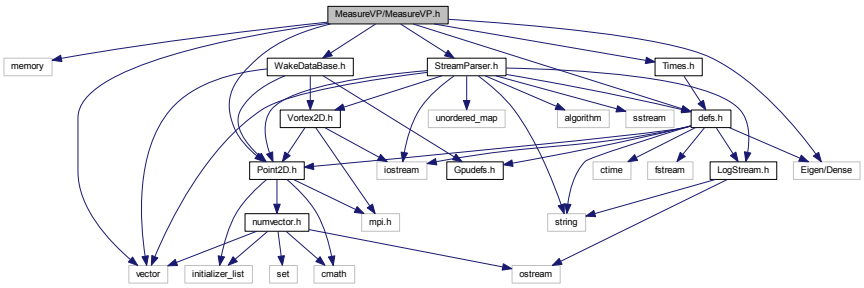
Дата

20 февраля 2019 г.

11.17 Файл MeasureVP/MeasureVP.h

Заголовочный файл с описанием класса MeasureVP.

#include <memory>
#include <vector>
#include "Eigen/Dense"
#include "defs.h"
#include "Point2D.h"
#include "StreamParser.h"
#include "Times.h"
#include "WakeDataBase.h"
Граф включаемых заголовочных файлов для MeasureVP.h:



Граф файлов, в которые включается этот файл:

Классы

- class MeasureVP

  Класс, отвечающий за вычисление поля скорости и давления в заданых точках для вывода

11.17.1 Подробное описание

Заголовочный файл с описанием класса MeasureVP.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

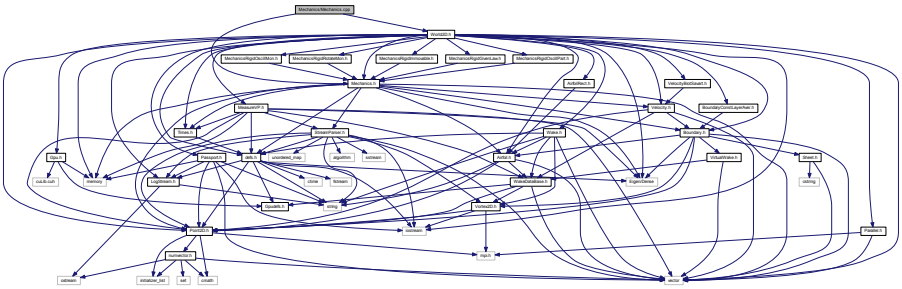
Дата

20 февраля 2019 г.

11.18 Файл Mechanics/Mechanics.cpp

Файл кода с описанием класса Mechanics.cpp.

#include "Mechanics.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для Mechanics.cpp:



11.18.1 Подробное описание

Файл кода с описанием класса Mechanics.cpp.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

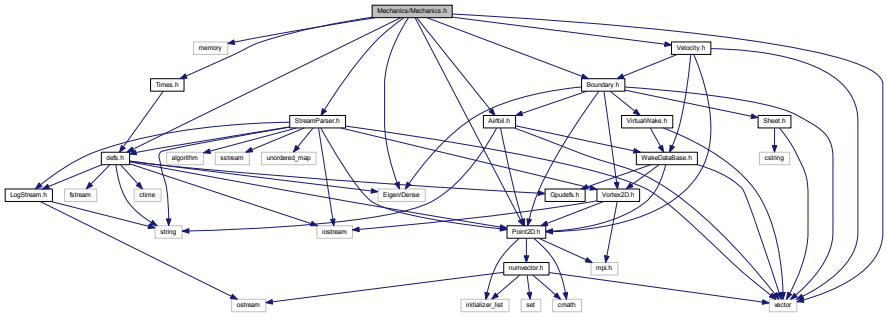
20 февраля 2019 г.

11.19 Файл Mechanics/Mechanics.h

Заголовочный файл с описанием класса Mechanics.

#include <memory>
#include <vector>
#include "Eigen/Dense"
#include "defs.h"
#include "Airfoil.h"
#include "Boundary.h"
#include "Point2D.h"
#include "StreamParser.h"
#include "Times.h"

#include "Velocity.h"
Граф включаемых заголовочных файлов для Mechanics.h:



Граф файлов, в которые включается этот файл:

Классы

- class Mechanics

  Абстрактный класс, определяющий вид механической системы

11.19.1 Подробное описание

Заголовочный файл с описанием класса Mechanics.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

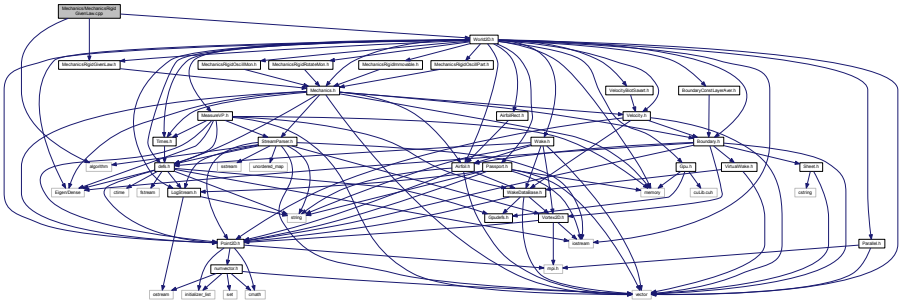
Дата

20 февраля 2019 г.

11.20 Файл Mechanics/MechanicsRigidGivenLaw.cpp

Файл кода с описанием класса MechanicsRigidGivenLaw.

#include <algorithm>
#include "MechanicsRigidGivenLaw.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для MechanicsRigidGivenLaw.cpp:



11.20.1 Подробное описание

Файл кода с описанием класса MechanicsRigidGivenLaw.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

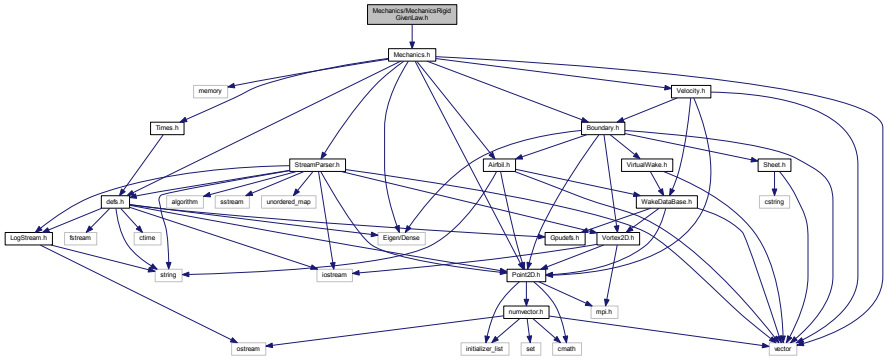
Дата

20 февраля 2019 г.

11.21 Файл Mechanics/MechanicsRigidGivenLaw.h

Заголовочный файл с описанием класса MechanicsRigidGivenLaw.

#include "Mechanics.h"
Граф включаемых заголовочных файлов для MechanicsRigidGivenLaw.h:



Граф файлов, в которые включается этот файл:

Классы

- class MechanicsRigidGivenLaw

  Класс, определяющий вид механической системы

11.21.1 Подробное описание

Заголовочный файл с описанием класса MechanicsRigidGivenLaw.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

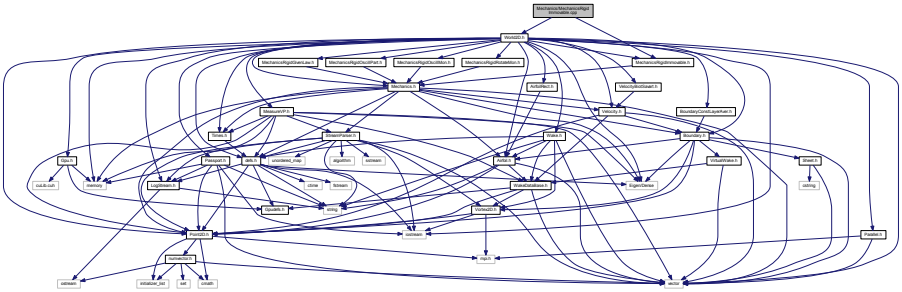
Дата

20 февраля 2019 г.

11.22 Файл Mechanics/MechanicsRigidImmovable.cpp

Файл кода с описанием класса MechanicsRigidImmovable.

#include "MechanicsRigidImmovable.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для MechanicsRigidImmovable.cpp:



11.22.1 Подробное описание

Файл кода с описанием класса MechanicsRigidImmovable.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

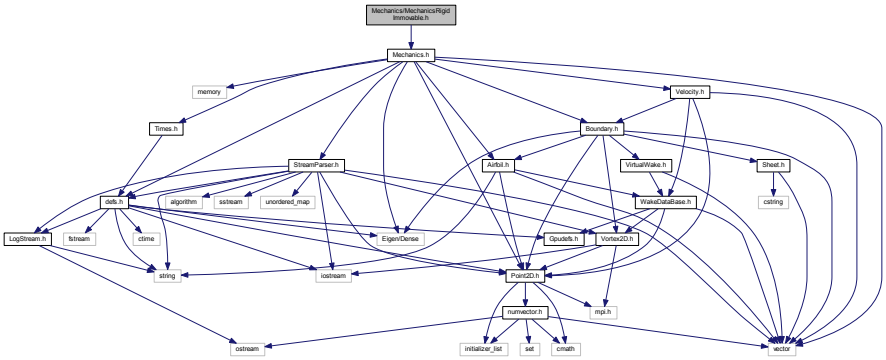
Дата

20 февраля 2019 г.

11.23 Файл Mechanics/MechanicsRigidImmovable.h

Заголовочный файл с описанием класса MechanicsRigidImmovable.

#include "Mechanics.h"
Граф включаемых заголовочных файлов для MechanicsRigidImmovable.h:



Граф файлов, в которые включается этот файл:

Классы

- class MechanicsRigidImmovable

  Класс, определяющий вид механической системы

11.23.1 Подробное описание

Заголовочный файл с описанием класса MechanicsRigidImmovable.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

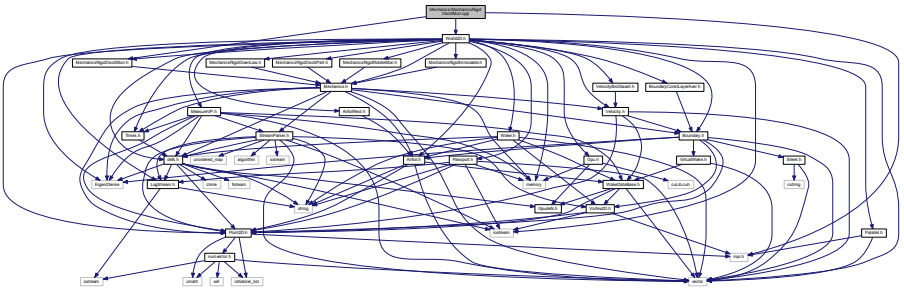
Дата

20 февраля 2019 г.

11.24 Файл Mechanics/MechanicsRigidOscillMon.cpp

Файл кода с описанием класса MechanicsRigidOscillMon.

#include "mpi.h"
#include "MechanicsRigidOscillMon.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для MechanicsRigidOscillMon.cpp:



11.24.1 Подробное описание

Файл кода с описанием класса MechanicsRigidOscillMon.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

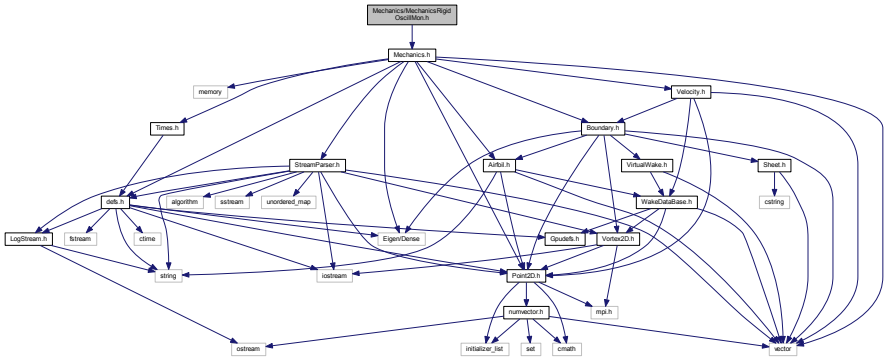
Дата

20 февраля 2019 г.

11.25 Файл Mechanics/MechanicsRigidOscillMon.h

Заголовочный файл с описанием класса MechanicsRigidOscillMon.

#include "Mechanics.h"
Граф включаемых заголовочных файлов для MechanicsRigidOscillMon.h:



Граф файлов, в которые включается этот файл:

Классы

- class MechanicsRigidOscillMon

  Класс, определяющий вид механической системы

11.25.1 Подробное описание

Заголовочный файл с описанием класса MechanicsRigidOscillMon.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

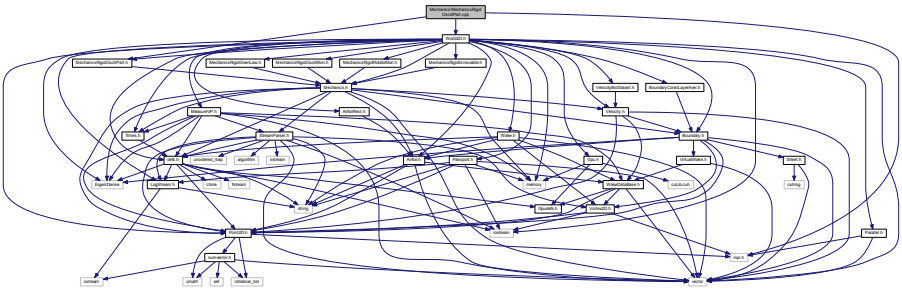
Дата

20 февраля 2019 г.

11.26 Файл Mechanics/MechanicsRigidOscillPart.cpp

Файл кода с описанием класса MechanicsRigidOscillPart.

#include "mpi.h"
#include "MechanicsRigidOscillPart.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для MechanicsRigidOscillPart.cpp:



11.26.1 Подробное описание

Файл кода с описанием класса MechanicsRigidOscillPart.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

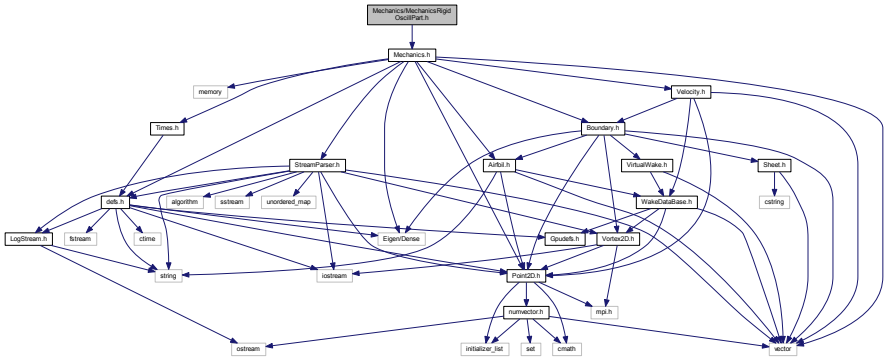
Дата

20 февраля 2019 г.

11.27 Файл Mechanics/MechanicsRigidOscillPart.h

Заголовочный файл с описанием класса MechanicsRigidOscillPart.

#include "Mechanics.h"
Граф включаемых заголовочных файлов для MechanicsRigidOscillPart.h:



Граф файлов, в которые включается этот файл:

Классы

- class MechanicsRigidOscillPart

  Класс, определяющий вид механической системы

11.27.1 Подробное описание

Заголовочный файл с описанием класса MechanicsRigidOscillPart.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

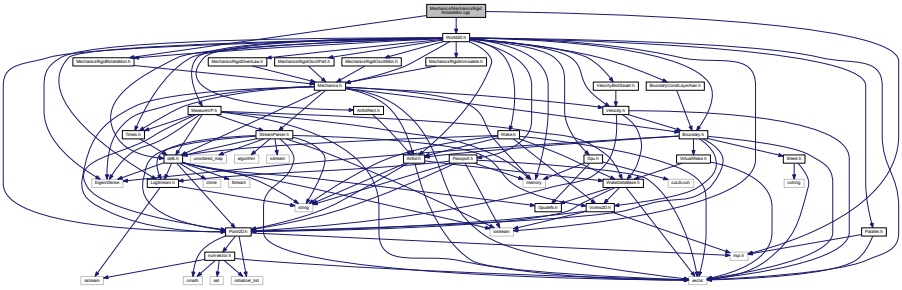
Дата

20 февраля 2019 г.

11.28 Файл Mechanics/MechanicsRigidRotateMon.cpp

Файл кода с описанием класса MechanicsRigidRotateMon.

#include "mpi.h"
#include "MechanicsRigidRotateMon.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для MechanicsRigidRotateMon.cpp:



11.28.1 Подробное описание

Файл кода с описанием класса MechanicsRigidRotateMon.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

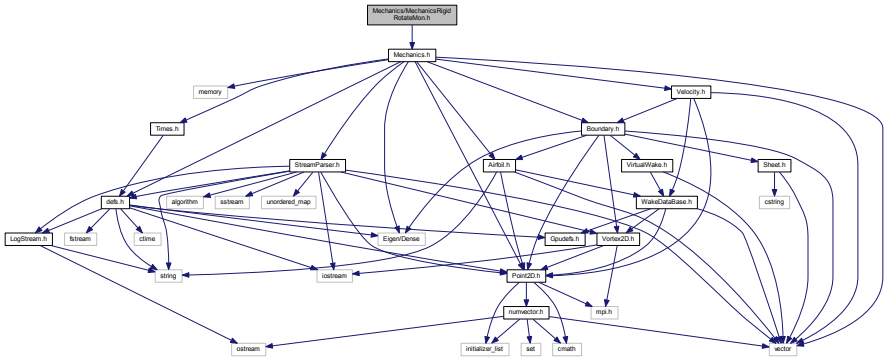
Дата

20 февраля 2019 г.

11.29 Файл Mechanics/MechanicsRigidRotateMon.h

Заголовочный файл с описанием класса MechanicsRigidRotateMon.

#include "Mechanics.h"
Граф включаемых заголовочных файлов для MechanicsRigidRotateMon.h:



Граф файлов, в которые включается этот файл:

Классы

- class MechanicsRigidRotateMon

  Класс, определяющий вид механической системы

11.29.1 Подробное описание

Заголовочный файл с описанием класса MechanicsRigidRotateMon.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

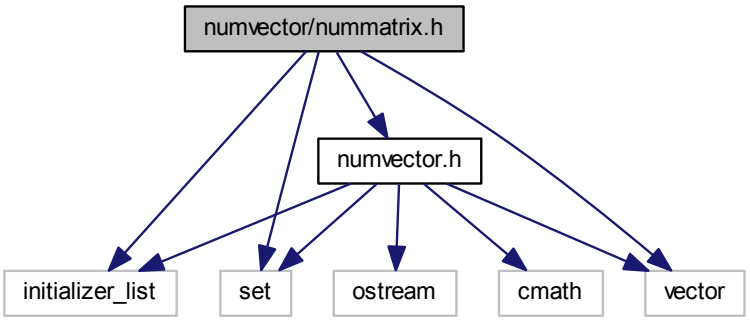
Дата

20 февраля 2019 г.

11.30 Файл numvector/nummatrix.h

Описание класса nummatrix.

#include <initializer_list>
#include <set>
#include <vector>
#include "numvector.h"
Граф включаемых заголовочных файлов для nummatrix.h:



Классы

- class nummatrix< T, n, m >

  Шаблонный класс, определяющий матрицу фиксированного размера
  Фактически представляет собой массив, для которого определено большое количество различных операций.
  Для доступа к элементам матрицы используется оператор [][]
  Оператор[] имитирует возврат ссылки/константной ссылки на numvector.

Функции

- template<typename T , typename P , size_t n, size_t m>
  auto operator & (const numvector< P, n > &x, const nummatrix< T, n, m > &A) -> numvector< std::remove_const< decltype(r[0] *x[0])>::type, m >

  Умножение вектора на матрицу
- template<typename T , typename P , size_t n, size_t m>
  auto operator| (const numvector< T, n > &x, const numvector< P, m > &y) -> nummatrix< std::remove_const< decltype(x[0] *y[0])>::type, n, m >

  Умножение вектора на вектор внешним образом
- template<typename T , size_t n, size_t m>
  std::ostream & operator<< (std::ostream &str, const nummatrix< T, n, m > &x)

  Перегрузка оператора "<<" вывода в поток

11.30.1 Подробное описание

Описание класса nummatrix.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

11.30.2 Функции

11.30.2.1 operator &()

```
template<typename T , typename P , size_t n, size_t m>
auto operator& (
            const numvector< P, n > & x,
            const nummatrix< T, n, m > & A ) -> numvector<std::remove_const<decltype(r[0] * x[0])>::type, m>
```

Умножение вектора на матрицу

Template Parameters

T	тип данных матрицы
P	тип данных вектора
n	длина вектора и число строк матрицы
m	число столбцов матрицы

Аргументы

in	A	константная ссылка на матрицу
in	x	константная ссылка на вектор

Возвращает

вектор — результат умножения матрицы на вектор

См. определение в файле nummatrix.h строка 523

```
525 {
526     numvector<std::remove_const<decltype(r[0] * x[0])>::type
    , m> res;
527     for (size_t i = 0; i < m; ++i)
528     {
529         res[i] = 0;
530         for (size_t j = 0; j < n; ++j)
531             res[i] += x[j] * A[j][i];
532     }
533     return res;
534 }//operator&(...)
```

11.30.2.2 operator<<()

```
template<typename T , size_t n, size_t m>
std::ostream& operator<< (
                std::ostream & str,
                const nummatrix< T, n, m > & x )
```

Перегрузка оператора "<<" вывода в поток

Выводит в поток вектор, элементы которого записаны в фигурных скобках и разделены запятой с пробелом

Template Parameters

T	тип данных
n	размерность вектора

Аргументы

in,out	str	ссылка на поток вывода

Аргументы

in	x	константная ссылка на вектор

Возвращает

ссылка на поток вывода

См. определение в файле nummatrix.h строка 563

```
{
    str << "{ ";
    for (size_t j = 0; j < n - 1; ++j)
        str << x[j] << ", ";
    str << x[n - 1];
    str << " }";
    return str;
}//operator<<
```

11.30.2.3 operator()

template<typename T , typename P , size_t n, size_t m>
auto operator| (
const numvector< T, n > & x,
const numvector< P, m > & y) -> nummatrix<std::remove_const<decltype(x[0] ∗ y[0])>::type, n, m>
[inline]

Умножение вектора на вектор внешним образом

См. определение в файле nummatrix.h строка 539

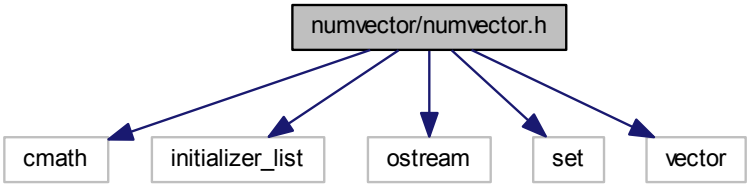
```
{
    nummatrix<std::remove_const<decltype(x[0] * y[0])>::type
, n, m> res;

    for (size_t i = 0; i < n; ++i)
    for (size_t j = 0; j < m; ++j)
        res[i][j] = x[i] * y[j];

    return res;
}//operator|(...)
```

11.31 Файл numvector/numvector.h

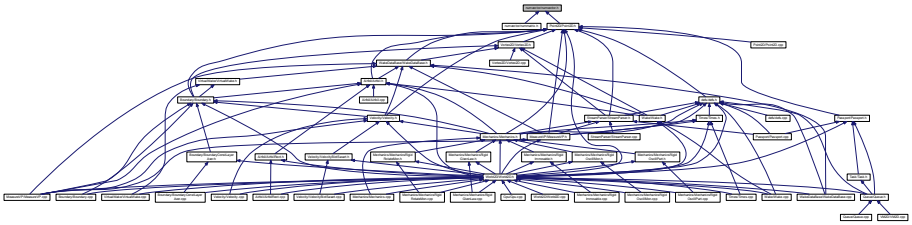
Описание класса numvector.

#include <cmath>
#include <initializer_list>
#include <ostream>
#include <set>

#include <vector>
Граф включаемых заголовочных файлов для numvector.h:



Граф файлов, в которые включается этот файл:



Классы

- class numvector< T, n >

  Шаблонный класс, определяющий вектор фиксированной длины
  Фактически представляет собой массив, для которого определено большое количество различных операций.
  Для доступа к элементам массива используется оператор [].

Макросы

- #define DEPRECATED

Функции

- template<typename T , typename P , size_t n>
  auto operator∗ (const P c, const numvector< T, n > &x) -> numvector< typename std::remove↩ _const< decltype(x[0] ∗c)>::type, n >

  Оператор "∗" умножения вектора на число (число слева, вектор справа)
- template<typename T , typename P , typename R >
  void cross (const numvector< T, 3 > &x, const numvector< P, 3 > &y, numvector< R, 3 > &z)

  Быстрое вычисление векторного произведения
- template<typename T , typename P , size_t n>
  auto dist2 (const numvector< T, n > &x, const numvector< P, n > &y) -> typename std::remove↩ _const< decltype(x[0] - y[0])>::type

  Вычисление квадрата расстояния между двумя точками

- template<typename R = double, typename T , typename P , size_t n>
  R dist (const numvector< T, n > &x, const numvector< P, n > &y)

  Вычисление расстояния между двумя точками
- template<typename T , size_t n>
  std::ostream & operator<< (std::ostream &str, const numvector< T, n > &x)

  Оператор "<<" вывода вектора в поток
- template<typename T , typename P , typename R , typename S , size_t n>
  std::pair< numvector< T, n >, numvector< P, n > > & operator+= (std::pair< numvector< T, n >, numvector< P, n >> &a, const std::pair< numvector< R, n >, numvector< S, n >> &b)

  Оператор прибавления "+=" для пар векторов
- template<typename T , typename P , typename R , typename S , size_t n>
  auto operator+ (const std::pair< numvector< T, n >, numvector< P, n >> &a, const std::pair< numvector< R, n >, numvector< S, n >> &b) -> std::pair< numvector< typename std::remove↩_const< decltype(a.first[0]+b.first[0])>::type, n >, numvector< typename std::remove_const< decltype(a.second[0]+b.second[0])>::type, n >>

  Оператор сложения "+" для пар векторов
- template<typename T , typename P , typename R , size_t n>
  std::pair< numvector< T, n >, numvector< P, n > > & operator*= (std::pair< numvector< T, n >, numvector< P, n >> &a, R c)

  Оператор домножения "*=" пары векторов на число (пара слева, число справа)
- template<typename T , typename P , typename R , size_t n>
  auto operator* (R c, const std::pair< numvector< T, n >, numvector< P, n >> &a) -> std↩::pair< numvector< typename std::remove_const< decltype(c *a.first[0])>::type, n >, numvector< typename std::remove_const< decltype(c *a.second[0])>::type, n >>

  Оператор умножения "*" числа на пару векторов (число слева, пара справа)
- template<typename T , typename P , typename R , size_t n>
  auto operator* (const std::pair< numvector< T, n >, numvector< P, n >> &a, R c) -> std↩::pair< numvector< typename std::remove_const< decltype(a.first[0] *c)>::type, n >, numvector< typename std::remove_const< decltype(a.second[0] *c)>::type, n >>

  Оператор умножения "*" пары векторов на число (пара слева, число справа)
- template<typename T , typename P , size_t n>
  std::ostream & operator<< (std::ostream &str, const std::pair< numvector< T, n >, numvector< P, n >> &x)

  Оператор "<<" вывода пары векторов в поток
- template<typename T , size_t n>
  numvector< T, n > dot (const numvector< numvector< T, n >, n > &A, const numvector< T, n > &x)

  Умножение квадратной матрицы на вектор (без приведения типов)
- template<typename T , size_t n>
  numvector< T, n > dot (const numvector< T, n > &x, const numvector< numvector< T, n >, n > &A)

  Умножение вектора на квадратную матрицу (без приведения типов)
- template<typename T , size_t n>
  numvector< T, n > matDotVec (const numvector< numvector< T, n >, n > &A, const numvector< T, n > &x)

  Умножение квадратной матрицы на вектор
- template<typename T , size_t n>
  numvector< numvector< T, n >, n > KronProd (const numvector< T, n > &x, const numvector< T, n > &y)

  Умножение вектора на вектор той же размерности внешним образом
- template<typename T , size_t n>
  numvector< numvector< T, n >, n > Transpose (const numvector< numvector< T, n >, n > &A)

  Транспонирование квадратной матрицы
- template<typename T , typename P , size_t n>
  double cross3 (const numvector< T, n > &x, const numvector< P, n > &y)

Вычисление третьей компоненты векторного произведения

- double operator∗ (const Point2D &x, const Point2D &y)

  Скалярное умножение двух векторов
- double operator∗ (const numvector< double, 2 > &x, const Point2D &y)

  Скалярное умножение двух векторов
- double operator∗ (const Point2D &x, const numvector< double, 2 > &y)

  Скалярное умножение двух векторов
- double operator∗ (const Point &x, const Point &y)

  Скалярное умножение двух векторов
- double operator∗ (const numvector< double, 2 > &x, const Point &y)

  Скалярное умножение двух векторов
- double operator∗ (const Point &x, const numvector< double, 2 > &y)

  Скалярное умножение двух векторов
- double operator∗ (const numvector< double, 2 > &x, const numvector< double, 2 > &y)

  Скалярное умножение двух векторов
- double operator∗ (const numvector< double, 3 > &x, const numvector< double, 3 > &y)

  Скалярное умножение двух векторов

11.31.1 Подробное описание

Описание класса numvector.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

11.31.2 Макросы

11.31.2.1 DEPRECATED

#define DEPRECATED

См. определение в файле numvector.h строка 59

11.31.3 Функции

11.31.3.1 cross()

```
template<typename T , typename P , typename R >
void cross (
            const numvector< T, 3 > & x,
            const numvector< P, 3 > & y,
            numvector< R, 3 > & z )   [inline]
```

Быстрое вычисление векторного произведения

Определено только для трехмерных векторов
Оптимизировано за счет отсутствия вызова конструктора, предполагает наличие трех уже созданных векторов

Template Parameters

T	тип данных компонент вектора первого множителя
P	тип данных компонент вектора второго множителя
R	тип данных компонент вектора результата

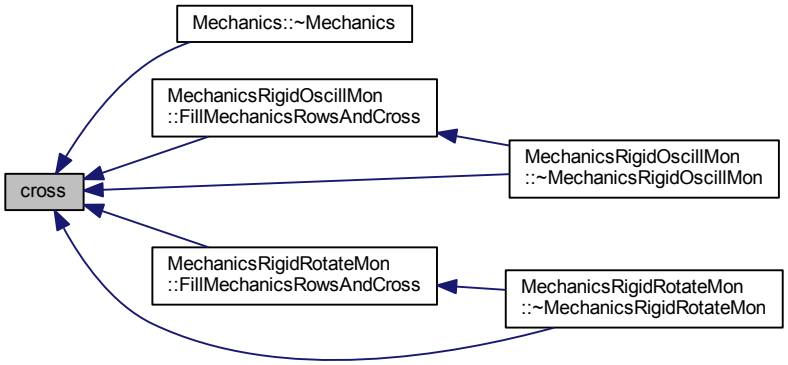
Аргументы

in	x	константная ссылка на первый множитель
in	y	константная ссылка на второй множитель
out	z	ссылка на результат векторного умножения

См. определение в файле numvector.h строка 738

```
{
    z = { x[1] * y[2] - x[2] * y[1], x[2] * y[0] - x[0] * y[2], x[0] * y[1] - x[1] * y[0] };
}//cross(...)
```

Граф вызова функции:



11.31.3.2 cross3()

```
template<typename T , typename P , size_t n>
double cross3 (
            const numvector< T, n > & x,
            const numvector< P, n > & y )   [inline]
```

Вычисление третьей компоненты векторного произведения

Определено для векторов любой размерности, используются только первые 2 компоненты исходных векторов

Template Parameters

T	тип данных компонент вектора - первого множителя
P	тип данных компонент вектора - второго множителя
n	размерность исходных векторов

Аргументы

in	x	константная ссылка на первый множитель
in	y	константная ссылка на второй множитель

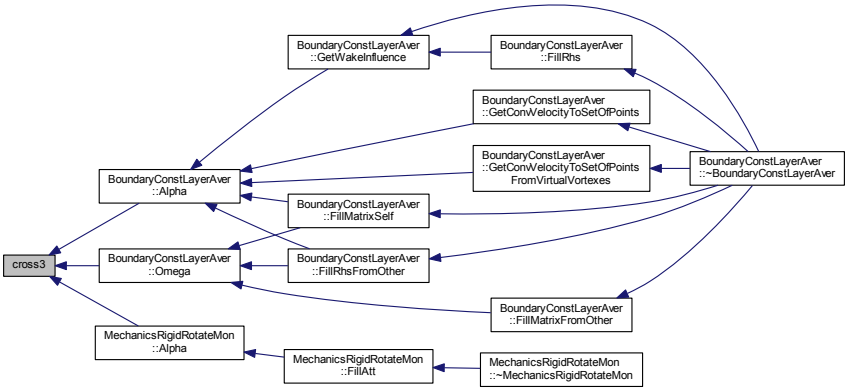
Возвращает

третья компонента вектороного произведения

См. определение в файле numvector.h строка 1031

```
{
    //deprecate: use numvector.operator^ instead of cross3(numvector<>,numvector<>)

    return (x[0] * y[1] - x[1] * y[0]);
}//cross3(...)
```

Граф вызова функции:



11.31.3.3 dist()

template<typename R = double, typename T , typename P , size_t n>
R dist (
const numvector< T, n > & x,
const numvector< P, n > & y) [inline]

Вычисление расстояния между двумя точками

Template Parameters

T	тип данных компонент радиус-вектора первой точки
P	тип данных компонент радиус-вектора второй точки
R	тип данных результата (по умолчанию double)

Аргументы

in	x	константная ссылка на радиус-вектор первой точки
in	y	константная ссылка на радиус-вектор второй точки

Возвращает

расстояние между точками

См. определение в файле numvector.h строка 770

```
{
    numvector<R, n> p = x - y;
    return sqrt(p&p);
}//dist(...)
```

11.31.3.4 dist2()

```
template<typename T , typename P , size_t n>
auto dist2 (
            const numvector< T, n > & x,
            const numvector< P, n > & y ) -> typename std::remove_const<decltype(x[0] - y[0])>::type    [inline]
```

Вычисление квадрата расстояния между двумя точками

Template Parameters

T	тип данных компонент первого радиус-вектора
P	тип данных компонент второго радиус-вектора
n	размерность векторов

Аргументы

in	x	константная ссылка на радиус-вектор первой точки
in	y	константная ссылка на радиус-вектор второй точки

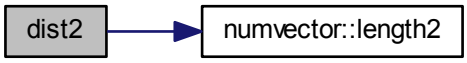
Возвращает

квадрат расстояния между точками

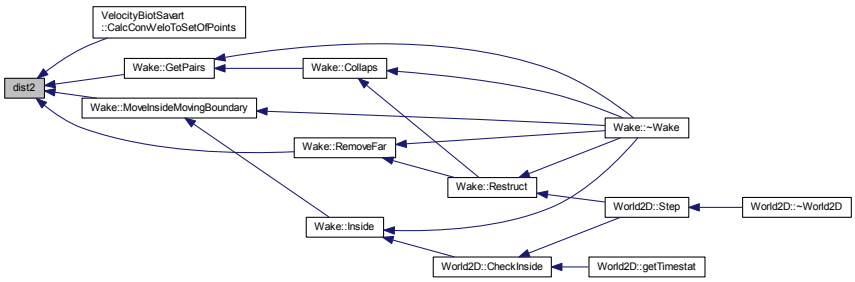
См. определение в файле numvector.h строка 754

```
{
    numvector<typename std::remove_const<decltype(x[0] - y[0])>::type, n> p = x - y;
    return p.length2();
}//dist2(...)
```

Граф вызовов:



Граф вызова функции:



11.31.3.5 dot() [1/2]

```
template<typename T , size_t n>
numvector<T, n> dot (
            const numvector< numvector< T, n >, n > & A,
            const numvector< T, n > & x )   [inline]
```

Умножение квадратной матрицы на вектор (без приведения типов)

Template Parameters

T	тип данных компонент матрицы и вектора
n	размерность матрицы и вектора

Аргументы

in	A	константная ссылка на матрицу
in	x	константная ссылка на вектор

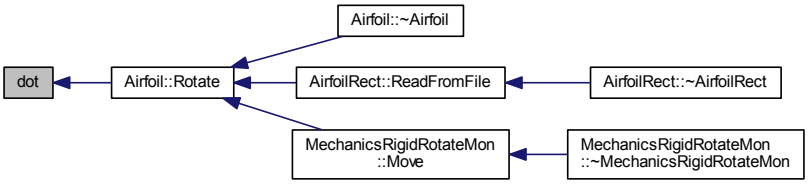
Возвращает

вектор результат умножения матрицы на вектор

См. определение в файле numvector.h строка 929

```
{
    //deprecate: use nummatrix.operator& instead of dot(numvector<numvector<>>, numvector<>)

    numvector<T, n> res;
    for (size_t i = 0; i < n; ++i)
        res[i] = A[i] & x;
    return res;
}//dot(...)
```

Граф вызова функции:



11.31.3.6 dot() [2/2]

```
template<typename T , size_t n>
numvector<T, n> dot (
            const numvector< T, n > & x,
            const numvector< numvector< T, n >, n > & A )   [inline]
```

Умножение вектора на квадратную матрицу (без приведения типов)

Template Parameters

T	тип данных компонент матрицы и вектора
n	размерность матрицы и вектора

Аргументы

in	A	константная ссылка на матрицу
in	x	константная ссылка на вектор

Возвращает

вектор результат умножения матрицы на вектор

См. определение в файле numvector.h строка 948

```
949 {
950     //deprecate: use operator&(numvector<>, nummatrix<>) instead of dot(numvector<>,
    numvector<numvector<>>)
951
952     numvector<T, n> res;
953     for (size_t i = 0; i < n; ++i)
954     {
955         res[i] = 0.0;
956         for (size_t j = 0; j < n; ++j)
957             res[i] += x[j] * A[j][i];
958     }
959     return res;
960 }//dot(...)
```

11.31.3.7 KronProd()

```
template<typename T , size_t n>
numvector<numvector<T, n>, n> KronProd (
            const numvector< T, n > & x,
            const numvector< T, n > & y )   [inline]
```

Умножение вектора на вектор той же размерности внешним образом

Template Parameters

T	тип данных компонент векторов
n	размерность векторов

Аргументы

in	x	константная ссылка на первый вектор
in	y	константная ссылка на второй вектор

Возвращает

матрицу ранга 1, являющюуся внешним (кронекеровым) произведением двух векторов

См. определение в файле numvector.h строка 990

```
991 {
992     //deprecate: use operator|(numvector<>, numvector<>) instead of KronProd(numvector<>, numvector<>)
993
994     numvector<numvector<T, n>, n> res;
995     for (size_t i = 0; i < n; ++i)
996         for (size_t j = 0; j < n; ++j)
997             res[i][j] = x[i] * y[j];
998     return res;
999 }//KronProd(...)
```

11.31.3.8 matDotVec()

```
template<typename T , size_t n>
numvector<T, n> matDotVec (
            const numvector< numvector< T, n >, n > & A,
            const numvector< T, n > & x )   [inline]
```

Умножение квадратной матрицы на вектор

Template Parameters

T	тип данных вектора и матрицы
n	длина вектора и размерность матрицы

Аргументы

in	A	константная ссылка на матрицу
in	x	константная ссылка на вектор

Возвращает

вектор — результат умножения матрицы на вектор

См. определение в файле numvector.h строка 971

```
{
    //deprecate: use nummatrix.operator& instead of matDotVec(numvector<numvector<>>, numvector<>)

    numvector<T, n> res;
    for (size_t i = 0; i < n; ++i)
        res[i] = A[i] * x;
    return res;
}//dot(...)
```

11.31.3.9 operator∗() [1/11]

```
template<typename T , typename P , size_t n>
auto operator∗ (
                const P c,
                const numvector< T, n > & x ) -> numvector<typename std::remove_const<decltype(x[0] ∗ c)>::type,
n>
```

Оператор "∗" умножения вектора на число (число слева, вектор справа)

Template Parameters

T	тип данных компонент вектора
P	тип данных числового множителя
n	длина вектора

Аргументы

in	c	числовой множитель
in	x	константная ссылка на умножаемый вектор

Возвращает

результат умножения вектора на число, приведенный к соответствующему типу

См. определение в файле numvector.h строка 716

```
{
```

```
    numvector<typename std::remove_const<decltype(x[0] * c)>::type, n> res;
    for (size_t i = 0; i < n; ++i)
        res[i] = x[i] * c;
    return res;
}//operator*(...)
```

11.31.3.10 operator∗() [2/11]

```
template<typename T , typename P , typename R , size_t n>
auto operator∗ (
                R c,
                const std::pair< numvector< T, n >, numvector< P, n >> & a ) -> std::pair<numvector<typename std::remove_const<decltype(c ∗ a.first[0])>::type, n>, numvector<typename std::remove_const<decltype(c ∗ a.second[0])>::type, n >>  [inline]
```

Оператор умножения "∗" числа на пару векторов (число слева, пара справа)

tparam T тип данных вектора - первого компонента в паре tparam P тип данных вектора - второго компонента в паре tparam R тип данных числового множителя tparam n длина векторов - компонентов пары param[in] c числовой множитель param[in] a константная ссылка на пару return пару, получаемую при покомпонентном умножении пары на число, приведенную к нужному типу

См. определение в файле numvector.h строка 871

```
{
    std::pair<numvector<typename std::remove_const<decltype(c * a.first[0])>::type, n>, numvector<typename std::remove_const<decltype(c * a.second[0])>::type, n >> res;
    res.first = c * a.first;
    res.second  = c * a.second;
    return res;
}//operator*(...)
```

11.31.3.11 operator∗() [3/11]

```
template<typename T , typename P , typename R , size_t n>
auto operator∗ (
                const std::pair< numvector< T, n >, numvector< P, n >> & a,
                R c ) -> std::pair<numvector<typename std::remove_const<decltype(a.first[0] ∗ c)>::type, n>, numvector<typename std::remove_const<decltype(a.second[0] ∗ c)>::type, n >>  [inline]
```

Оператор умножения "∗" пары векторов на число (пара слева, число справа)

tparam T тип данных вектора - первого компонента в паре tparam P тип данных вектора - второго компонента в паре tparam R тип данных числового множителя tparam n длина векторов - компонентов пары param[in] c числовой множитель param[in] a константная ссылка на пару return пару, получаемую при покомпонентном умножении пары на число, приведенную к нужному типу

См. определение в файле numvector.h строка 891

```
{
    return c * a;
}//operator*(...)
```

11.31.3.12 operator∗() [4/11]

```
double operator∗ (
                const Point2D & x,
                const Point2D & y )  [inline]
```

Скалярное умножение двух векторов

Аргументы

in	x	константная ссылка на первый множитель
in	y	константная ссылка на второй множитель

Возвращает

результат скалярного умножения

См. определение в файле numvector.h строка 1044

```
1045 {
1046     //deprecate: use numvector.operator& instead of operator*
1047
1048     const numvector<double, 2> xx = *(reinterpret_cast<const
     numvector<double, 2>*>(&x));
1049     const numvector<double, 2> yy = *(reinterpret_cast<const
     numvector<double, 2>*>(&y));
1050     return (xx & yy);
1051 }//operator*(...)
```

11.31.3.13 operator∗() [5/11]

double operator∗ (
const numvector< double, 2 > & x,
const Point2D & y) [inline]

Скалярное умножение двух векторов

Аргументы

in	x	константная ссылка на первый множитель
in	y	константная ссылка на второй множитель

Возвращает

результат скалярного умножения

См. определение в файле numvector.h строка 1059

```
1060 {
1061     //deprecate: use numvector.operator& instead of operator*
1062
1063     const numvector<double, 2> yy = *(reinterpret_cast<const
     numvector<double, 2>*>(&y));
1064     return (x & yy);
1065 }//operator*(...)
```

11.31.3.14 operator∗() [6/11]

```
double operator∗ (
            const Point2D & x,
            const numvector< double, 2 > & y )   [inline]
```

Скалярное умножение двух векторов

Аргументы

in	x	константная ссылка на первый множитель
in	y	константная ссылка на второй множитель

Возвращает

результат скалярного умножения

См. определение в файле numvector.h строка 1073

```
1074 {
1075     //deprecate: use numvector.operator& instead of operator*
1076
1077     const numvector<double, 2> xx = *(reinterpret_cast<const
     numvector<double, 2>*>(&x));
1078     return (xx & y);
1079 }//operator*(...)
```

11.31.3.15 operator∗() [7/11]

```
double operator∗ (
            const Point & x,
            const Point & y )   [inline]
```

Скалярное умножение двух векторов

Аргументы

in	x	константная ссылка на первый множитель
in	y	константная ссылка на второй множитель

Возвращает

результат скалярного умножения

См. определение в файле numvector.h строка 1087

```
1088 {
1089     //deprecate: use numvector.operator& instead of operator*
1090
1091     const numvector<double, 2> xx = *(reinterpret_cast<const
```

```
numvector<double, 2>*>(&x));
1092    const numvector<double, 2> yy = *(reinterpret_cast<const
numvector<double, 2>*>(&y));
1093    return (xx & yy);
1094 }//operator*(...)
```

11.31.3.16 operator∗() [8/11]

```
double operator∗ (
            const numvector< double, 2 > & x,
            const Point & y )   [inline]
```

Скалярное умножение двух векторов

Аргументы

in	x	константная ссылка на первый множитель
in	y	константная ссылка на второй множитель

Возвращает

результат скалярного умножения

См. определение в файле numvector.h строка 1102

```
1103 {
1104    //deprecate: use numvector.operator& instead of operator*
1105
1106    const numvector<double, 2> yy = *(reinterpret_cast<const
numvector<double, 2>*>(&y));
1107    return (x & yy);
1108 }//operator*(...)
```

11.31.3.17 operator∗() [9/11]

```
double operator∗ (
            const Point & x,
            const numvector< double, 2 > & y )   [inline]
```

Скалярное умножение двух векторов

Аргументы

in	x	константная ссылка на первый множитель
in	y	константная ссылка на второй множитель

Возвращает

результат скалярного умножения

См. определение в файле numvector.h строка 1116

```
{
    //deprecate: use numvector.operator& instead of operator*

    const numvector<double, 2> xx = *(reinterpret_cast<const numvector<double, 2>*>(&x));
    return (xx & y);
}//operator*(...)
```

11.31.3.18 operator∗() [10/11]

double operator∗ (
const numvector< double, 2 > & x,
const numvector< double, 2 > & y) [inline]

Скалярное умножение двух векторов

Аргументы

in	x	константная ссылка на первый множитель
in	y	константная ссылка на второй множитель

Возвращает

результат скалярного умножения

См. определение в файле numvector.h строка 1132

```
{
    //deprecate: use numvector.operator& instead of operator*

    return (x & y);
}//operator*(...)
```

11.31.3.19 operator∗() [11/11]

double operator∗ (
const numvector< double, 3 > & x,
const numvector< double, 3 > & y) [inline]

Скалярное умножение двух векторов

Аргументы

in	x	константная ссылка на первый множитель
in	y	константная ссылка на второй множитель

Возвращает

результат скалярного умножения

См. определение в файле numvector.h строка 1145

```
1146 {
1147     //deprecate: use numvector.operator& instead of operator*
1148
1149     return (x & y);
1150 }//operator*(...)
```

11.31.3.20 operator∗=()

```
template<typename T , typename P , typename R , size_t n>
std::pair<numvector<T, n>, numvector<P, n> >& operator∗= (
                std::pair< numvector< T, n >, numvector< P, n >> & a,
                R c )   [inline]
```

Оператор домножения "∗=" пары векторов на число (пара слева, число справа)

tparam T тип данных вектора - первого компонента в паре tparam P тип данных вектора - второго компонента в паре tparam R тип данных числового множителя tparam n длина векторов - компонентов пары param[in] a ссылка на пару param[in] c числовой множитель return ссылка на пару после ее покомпонентного домножения на число

См. определение в файле numvector.h строка 853

```
854 {
855     a.first *= c;
856     a.second *= c;
857     return a;
858 }//operator*=(...)
```

11.31.3.21 operator+()

```
template<typename T , typename P , typename R , typename S , size_t n>
auto operator+ (
            const std::pair< numvector< T, n >, numvector< P, n >> & a,
            const std::pair< numvector< R, n >, numvector< S, n >> & b ) -> std::pair<numvector<typename std↩
::remove_const<decltype(a.first[0] + b.first[0])>::type, n>, numvector<typename std::remove_const<decltype(a.second[0]
+ b.second[0])>::type, n >>    [inline]
```

Оператор сложения "+" для пар векторов

tparam T тип данных вектора - первого компонента в левой паре tparam P тип данных вектора - второго компонента в левой паре tparam R тип данных вектора - первого компонента в правой паре tparam S тип данных вектора - второго компонента в правой паре tparam n длина векторов - компонентов обеих пар param[in] a константная ссылка на левую пару param[in] b константная ссылка на правую пару return пару, получаемую при покомпонентном суммировании левой и правой пар, приведенную к нужному типу

См. определение в файле numvector.h строка 833

```
835 {
836     std::pair<numvector<typename std::remove_const<decltype(a.first[0] + b.first[0])>::type, n>,
    numvector<typename std::remove_const<decltype(a.second[0] + b.second[0])>::type, n >> res;
837     res.first = a.first + b.first;
838     res.second = a.second + b.second;
839     return res;
840 }//operator+(...)
```

11.31.3.22 operator+=()

```
template<typename T , typename P , typename R , typename S , size_t n>
std::pair<numvector<T, n>, numvector<P, n> >& operator+= (
            std::pair< numvector< T, n >, numvector< P, n >> & a,
            const std::pair< numvector< R, n >, numvector< S, n >> & b )    [inline]
```

Оператор прибавления "+=" для пар векторов

tparam T тип данных вектора - первого компонента в паре tparam P тип данных вектора - второго компонента в паре tparam R тип данных вектора - первого компонента в прибавляемой паре tparam S тип данных вектора - второго компонента в прибавляемой паре tparam n длина векторов - компонентов обеих пар param[in] a ссылка на первую пару param[in] b константная ссылка на прибавляемую пару return ссылку на первую пару после прибавления к ней второй пары

См. определение в файле numvector.h строка 814

```
815 {
816     a.first += b.first;
817     a.second += b.second;
818     return a;
819 }//operator+=(...)
```

11.31.3.23 operator<<() [1/2]

```
template<typename T , size_t n>
std::ostream& operator<< (
            std::ostream & str,
            const numvector< T, n > & x )
```

Оператор "<<" вывода вектора в поток

Выводит в поток вектор, элементы которого записываются в фигурных скобках и разделены запятой с пробелом

Template Parameters

T	тип данных компонент вектора
n	размерность вектора

Аргументы

in,out	str	ссылка на поток вывода
in	x	константная ссылка на вектор

Возвращает

ссылку на поток вывода

См. определение в файле numvector.h строка 787

```
788 {
789     str << "{ ";
790     for (size_t j = 0; j < n - 1; ++j)
791         str << x[j] << ", ";
792     str << x[n - 1];
793     str << " }";
794     return str;
795 }//operator<<(...)
```

11.31.3.24 operator<<() [2/2]

```
template<typename T , typename P , size_t n>
std::ostream& operator<< (
                std::ostream & str,
                const std::pair< numvector< T, n >, numvector< P, n >> & x )
```

Оператор "<<" вывода пары векторов в поток

Выводит в поток векторы из пары, разделяя их запятой с пробелом

Template Parameters

T	тип данных вектора - первого компонента пары
P	тип данных вектора - второго компонента пары
n	размерность векторов

Аргументы

in,out	str	ссылка на поток вывода
in	x	константная ссылка на вектор

Возвращает

ссылку на поток вывода

См. определение в файле numvector.h строка 909

```
910 {
911     str << "{ " << x.first << ", " << x.second << " }";
912     return str;
913 }//operator<<
```

11.31.3.25 Transpose()

```
template<typename T , size_t n>
numvector<numvector<T, n>, n> Transpose (
            const numvector< numvector< T, n >, n > & A )   [inline]
```

Транспонирование квадратной матрицы

Template Parameters

T	тип данных компонент матрицы
n	размерность матрицы

Возвращает

транспонированную матрицу

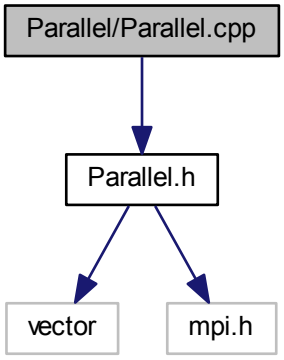
См. определение в файле numvector.h строка 1008

```
1009 {
1010     //deprecate: use member class nummatrix.transpose() instead of Transpose(numvector<numvector>)
1011
1012     numvector<numvector<T, n>, n> res;
1013     for (size_t i = 0; i < n; ++i)
1014         for (size_t j = 0; j < n; ++j)
1015             res[i][j] = A[j][i];
1016     return res;
1017 }
```

11.32 Файл Parallel/Parallel.cpp

Файл кода с описанием класса Parallel.

#include "Parallel.h"
Граф включаемых заголовочных файлов для Parallel.cpp:



11.32.1 Подробное описание

Файл кода с описанием класса Parallel.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

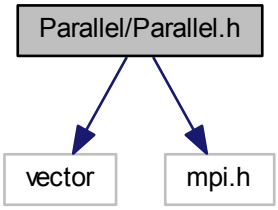
Дата

20 февраля 2019 г.

11.33 Файл Parallel/Parallel.h

Заголовочный файл с описанием класса Parallel и структуры parProp.

#include <vector>
#include "mpi.h"
Граф включаемых заголовочных файлов для Parallel.h:



Граф файлов, в которые включается этот файл:

Классы

- struct parProp

  Стрктура, содержащая параметры исполнения задачи в параллельном MPI-режиме
- class Parallel

  Класс, опеделяющий параметры исполнения задачи в параллельном MPI-режиме

11.33.1 Подробное описание

Заголовочный файл с описанием класса Parallel и структуры parProp.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

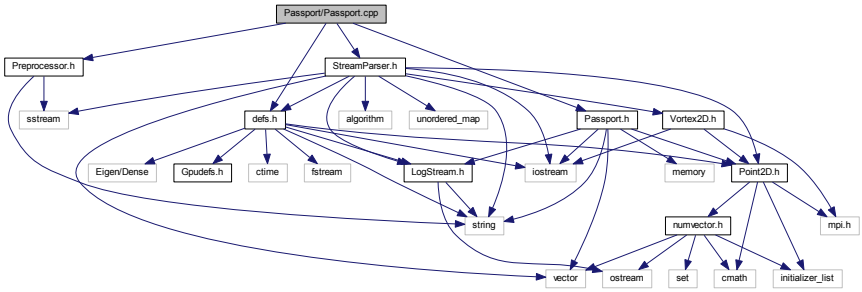
Дата

20 февраля 2019 г.

11.34 Файл Passport/Passport.cpp

Файл кода с описанием класса Passport.

#include "defs.h"
#include "Passport.h"
#include "Preprocessor.h"
#include "StreamParser.h"
Граф включаемых заголовочных файлов для Passport.cpp:



11.34.1 Подробное описание

Файл кода с описанием класса Passport.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

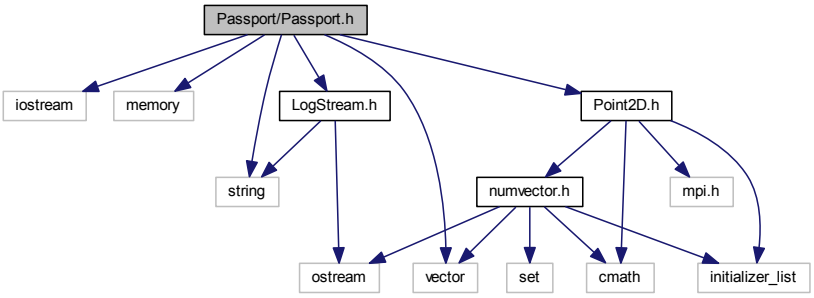
20 февраля 2019 г.

11.35 Файл Passport/Passport.h

Заголовочный файл с описанием класса Passport и соответствующими структурами

#include <iostream>
#include <memory>
#include <string>
#include <vector>
#include "Point2D.h"

#include "LogStream.h"
Граф включаемых заголовочных файлов для Passport.h:



Граф файлов, в которые включается этот файл:

Классы

- struct PhysicalProperties

  Структура, задающая физические свойства задачи
- struct TimeDiscretizationProperties

  Структура, задающая параметры процесса интегрирования по времени
- struct WakeDiscretizationProperties

  Структура, задающая параметры параметры дискретизации вихревого следа
- struct NumericalSchemes

  Структура, задающая используемые численные схемы
- struct AirfoilParams

  Структура, задающая параметры профиля
- class Passport

  Класс, опеделяющий паспорт задачи

11.35.1 Подробное описание

Заголовочный файл с описанием класса Passport и соответствующими структурами

Помимо класса Passport определены следующие структуры:

- PhysicalProperties — физические свойства задачи
- TimeDiscretizationProperties — параметры процесса интегрирования по времени
- WakeDiscretizationProperties — параметры дискретизации вихревого следа
- NumericalSchemes — используемые численные схемы
- AirfoilParams — параметры профиля

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

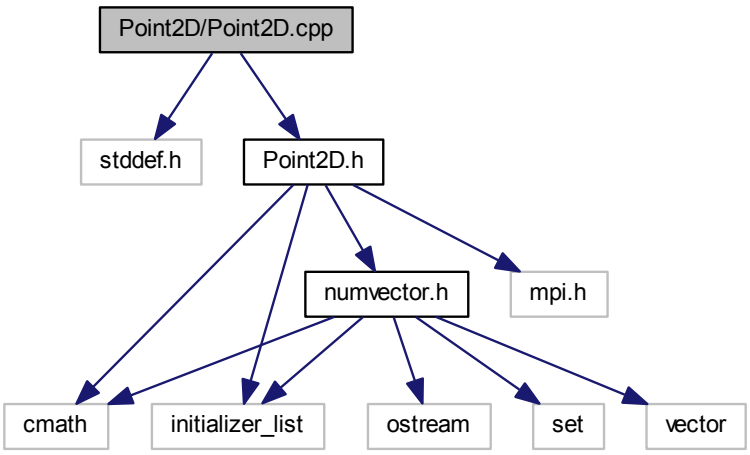
Дата

20 февраля 2019 г.

11.36 Файл Point2D/Point2D.cpp

Файл кода с описанием класса Point2D.

#include <stddef.h>
#include "Point2D.h"
Граф включаемых заголовочных файлов для Point2D.cpp:



11.36.1 Подробное описание

Файл кода с описанием класса Point2D.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

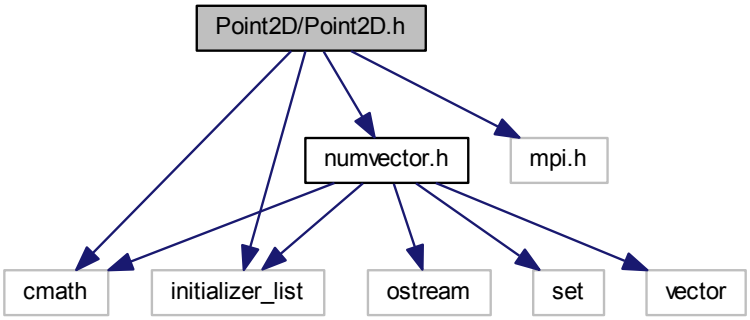
Дата

20 февраля 2019 г.

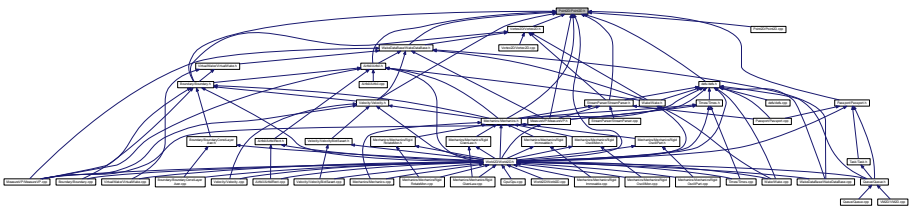
11.37 Файл Point2D/Point2D.h

Заголовочный файл с описанием класса Point2D.

#include <cmath>
#include <initializer_list>
#include "mpi.h"
#include "numvector.h"
Граф включаемых заголовочных файлов для Point2D.h:



Граф файлов, в которые включается этот файл:



Классы

- class Point2D

  Класс, опеделяющий двумерный вектор

11.37.1 Подробное описание

Заголовочный файл с описанием класса Point2D.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

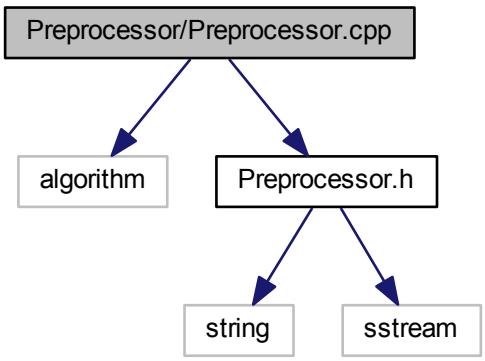
Дата

20 февраля 2019 г.

11.38 Файл Preprocessor/Preprocessor.cpp

Файл кода с описанием класса Preprocessor.

#include <algorithm>
#include "Preprocessor.h"
Граф включаемых заголовочных файлов для Preprocessor.cpp:



11.38.1 Подробное описание

Файл кода с описанием класса Preprocessor.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

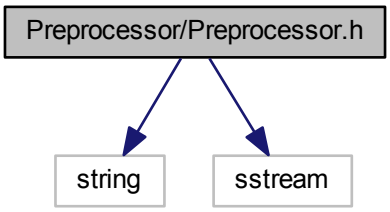
Дата

20 февраля 2019 г.

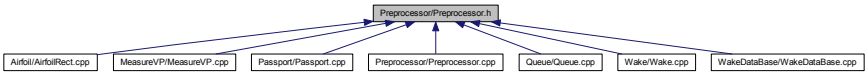
11.39 Файл Preprocessor/Preprocessor.h

Заголовочный файл с описанием класса Preprocessor.

#include <string>
#include <sstream>
Граф включаемых заголовочных файлов для Preprocessor.h:



Граф файлов, в которые включается этот файл:



Классы

- class Preprocessor

  Класс, позволяющий выполнять предварительную обработку файлов

11.39.1 Подробное описание

Заголовочный файл с описанием класса Preprocessor.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

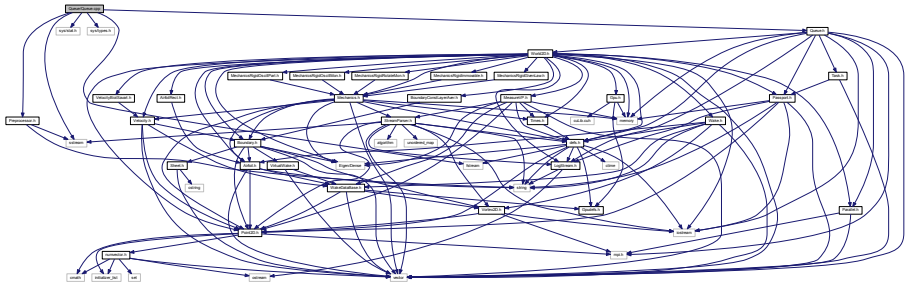
Дата

20 февраля 2019 г.

11.40 Файл Queue/Queue.cpp

Файл кода с описанием класса Queue.

#include <fstream>
#include <sstream>
#include <sys/stat.h>
#include <sys/types.h>
#include "Preprocessor.h"
#include "Queue.h"
Граф включаемых заголовочных файлов для Queue.cpp:



11.40.1 Подробное описание

Файл кода с описанием класса Queue.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

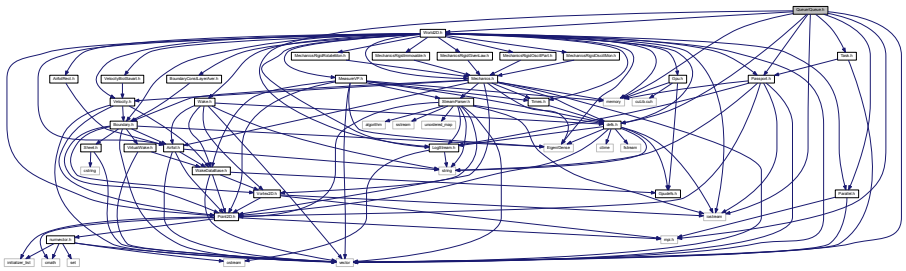
Дата

20 февраля 2019 г.

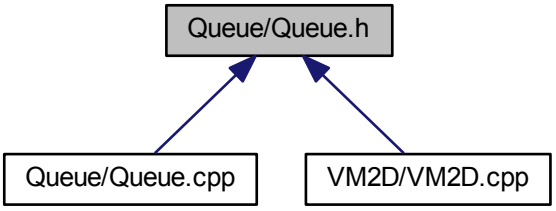
11.41 Файл Queue/Queue.h

Заголовочный файл с описанием класса Queue.

#include <iostream>
#include <memory>
#include <vector>
#include "mpi.h"
#include "defs.h"
#include "Parallel.h"
#include "Passport.h"
#include "LogStream.h"
#include "Task.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для Queue.h:



Граф файлов, в которые включается этот файл:



Классы

- class Queue

  Класс, опеделяющий список решаемых задач и очередь их прохождения

11.41.1 Подробное описание

Заголовочный файл с описанием класса Queue.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

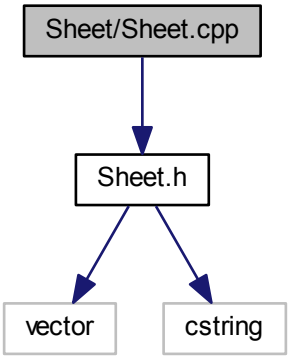
Дата

20 февраля 2019 г.

11.42 Файл Sheet/Sheet.cpp

Файл кода с описанием класса Sheet.

#include "Sheet.h"
Граф включаемых заголовочных файлов для Sheet.cpp:



11.42.1 Подробное описание

Файл кода с описанием класса Sheet.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

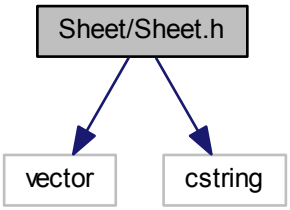
Дата

20 февраля 2019 г.

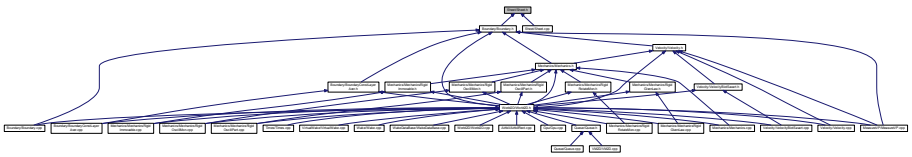
11.43 Файл Sheet/Sheet.h

Заголовочный файл с описанием класса Sheet.

#include <vector>
#include <cstring>
Граф включаемых заголовочных файлов для Sheet.h:



Граф файлов, в которые включается этот файл:



Классы

- class Sheet

  Класс, опеделяющий слои на поверхности обтекаемого профиля

11.43.1 Подробное описание

Заголовочный файл с описанием класса Sheet.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

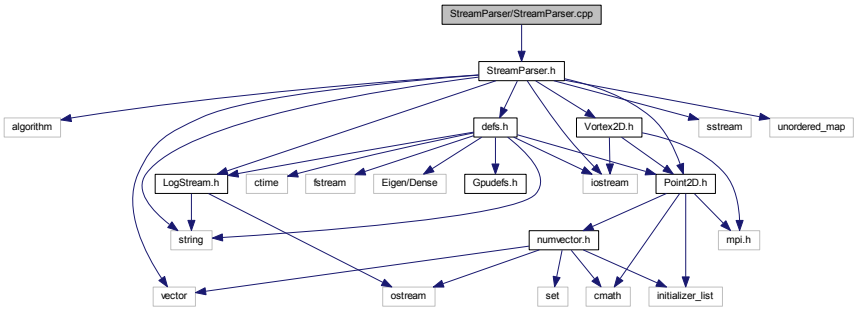
Дата

20 февраля 2019 г.

11.44 Файл StreamParser/StreamParser.cpp

Файл кода с описанием класса StreamParser.

#include "StreamParser.h"
Граф включаемых заголовочных файлов для StreamParser.cpp:



11.44.1 Подробное описание

Файл кода с описанием класса StreamParser.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

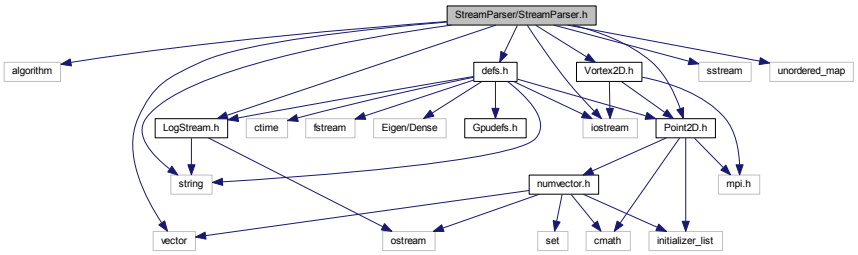
20 февраля 2019 г.

11.45 Файл StreamParser/StreamParser.h

Заголовочный файл с описанием класса StreamParser.

#include <algorithm>
#include <iostream>
#include <sstream>
#include <string>
#include <unordered_map>
#include <vector>
#include "defs.h"
#include "LogStream.h"

#include "Point2D.h"
#include "Vortex2D.h"
Граф включаемых заголовочных файлов для StreamParser.h:



Граф файлов, в которые включается этот файл:

Классы

- class StreamParser

  Класс, позволяющий выполнять разбор файлов и строк с настройками и параметрами

11.45.1 Подробное описание

Заголовочный файл с описанием класса StreamParser.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

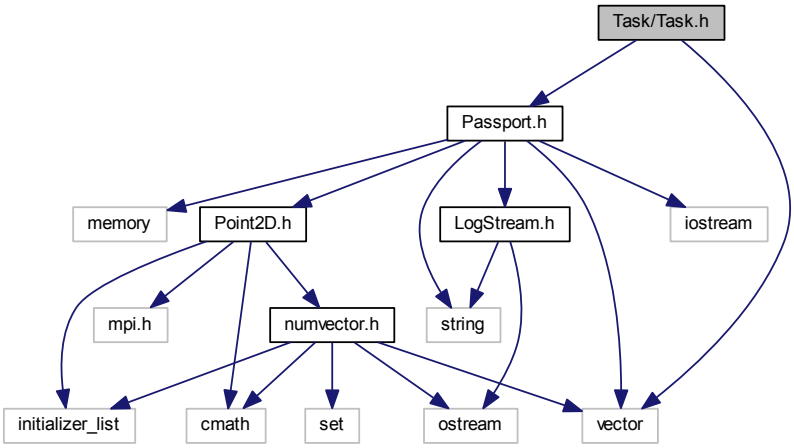
Дата

20 февраля 2019 г.

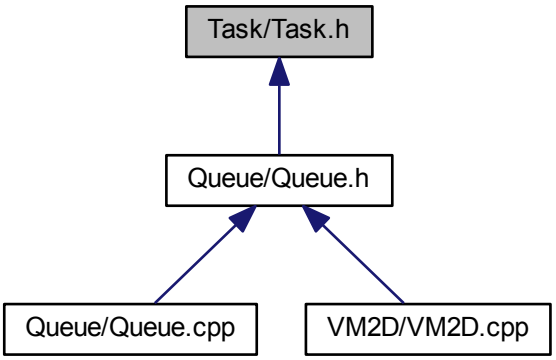
11.46 Файл Task/Task.h

Заголовочный файл с описанием класса Task.

#include <vector>
#include "Passport.h"
Граф включаемых заголовочных файлов для Task.h:



Граф файлов, в которые включается этот файл:



Классы

- class Task

  Класс, опеделяющий описание каждой задачи в смысле прохождения процесса ее решения

Перечисления

- enum TaskState {
  TaskState::waiting = 0, TaskState::starting = 1, TaskState::running = 2, TaskState::finishing = 3, TaskState::done = 4 }

  Тип-перечисление, опеделяющий состояние задачи в процессе работы программы

11.46.1 Подробное описание

Заголовочный файл с описанием класса Task.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

11.46.2 Перечисления

11.46.2.1 TaskState

enum TaskState [strong]

Тип-перечисление, опеделяющий состояние задачи в процессе работы программы

Может принимать 5 различных значений:

- waiting (0) — ожидает запуска;
- starting (1) — стартует;
- running (2) — считает;
- finishing (3) — останавливается;
- done (4) — отсчитано;

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

Элементы перечислений

waiting	задача ожидает запуска
starting	задача стартует
running	задача решается
finishing	задача финиширует
done	задача решена

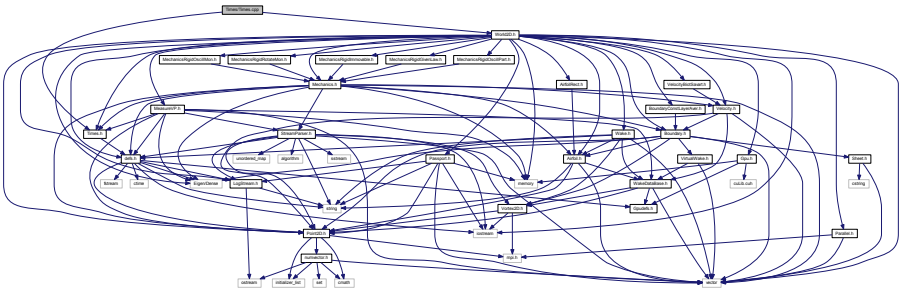
См. определение в файле Task.h строка 62

```
63 {
65     waiting   = 0,
66
68     starting  = 1,
69
71     running   = 2,
72
74     finishing = 3,
75
77     done      = 4
78 };
```

11.47 Файл Times/Times.cpp

Файл кода с описанием класса Times.

#include "Times.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для Times.cpp:



11.47.1 Подробное описание

Файл кода с описанием класса Times.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

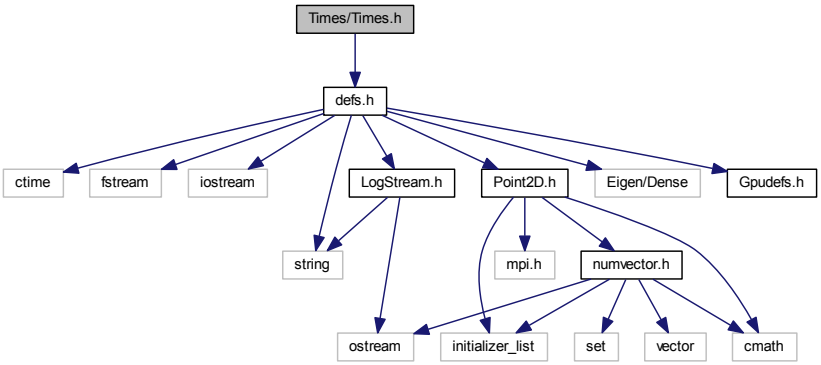
Дата

20 февраля 2019 г.

11.48 Файл Times/Times.h

Заголовочный файл с описанием класса Times.

#include "defs.h"
Граф включаемых заголовочных файлов для Times.h:



Граф файлов, в которые включается этот файл:

Классы

- class Times

  Класс для сбора статистики времени исполнения основных шагов алгоритма и вывода ее в файл

11.48.1 Подробное описание

Заголовочный файл с описанием класса Times.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

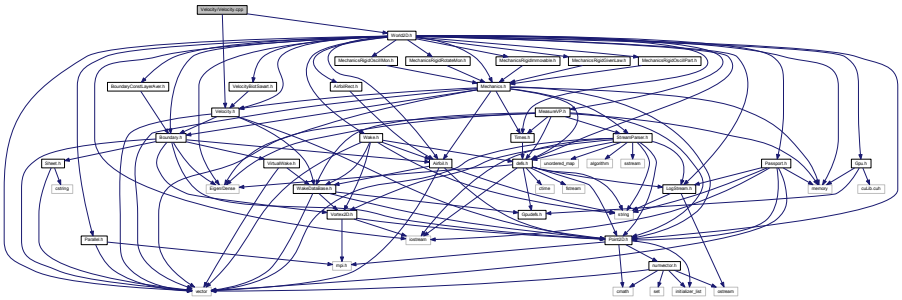
Дата

20 февраля 2019 г.

11.49 Файл Velocity/Velocity.cpp

Файл кода с описанием класса Velocity.

#include "Velocity.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для Velocity.cpp:



11.49.1 Подробное описание

Файл кода с описанием класса Velocity.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

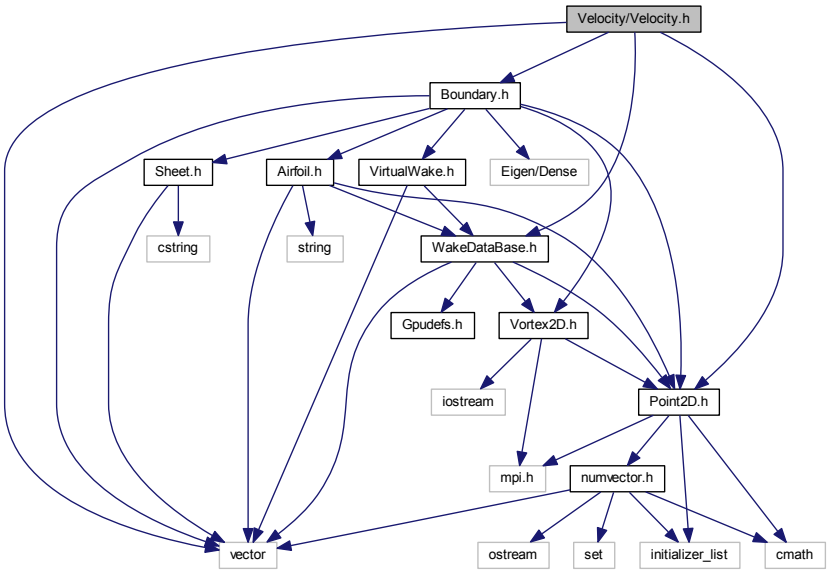
20 февраля 2019 г.

11.50 Файл Velocity/Velocity.h

Заголовочный файл с описанием класса Velocity.

#include <vector>
#include "Point2D.h"
#include "WakeDataBase.h"

#include "Boundary.h"
Граф включаемых заголовочных файлов для Velocity.h:



Граф файлов, в которые включается этот файл:

Классы

- struct VortexesParams

  Структура, определяющая параметры виртуальных вихрей для отдельного профиля
- class Velocity

  Абстрактный класс, определяющий способ вычисления скоростей

11.50.1 Подробное описание

Заголовочный файл с описанием класса Velocity.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

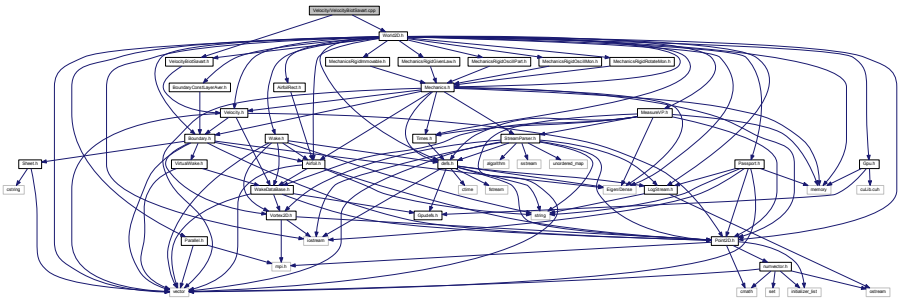
Дата

20 февраля 2019 г.

11.51 Файл Velocity/VelocityBiotSavart.cpp

Файл кода с описанием класса VelocityBiotSavart.

#include "VelocityBiotSavart.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для VelocityBiotSavart.cpp:



Функции

- void ModifyE2 (double *ee2, double dst2)

11.51.1 Подробное описание

Файл кода с описанием класса VelocityBiotSavart.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

11.51.2 Функции

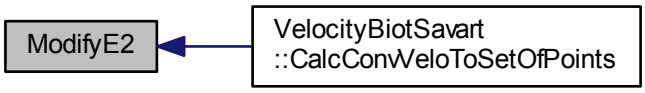
11.51.2.1 ModifyE2()

```
void ModifyE2 (
            double * ee2,
            double dst2 )   [inline]
```

См. определение в файле VelocityBiotSavart.cpp строка 55

```
56 {
57     if (dst2 > 0)
58     {
59         if (dst2<ee2[0])
60         {
61             ee2[2] = ee2[1];
62             ee2[1] = ee2[0];
63             ee2[0] = dst2;
64         }//if (dist2<ee2[0])
65         else
66         {
67             if (dst2<ee2[1])
68             {
69                 ee2[2] = ee2[1];
70                 ee2[1] = dst2;
71             }// if (dist2<ee2[1])
72             else
73             if (dst2<ee2[2])
74                 ee2[2] = dst2;
75         }//else
76     }//if (dst2>0)
77 }
```

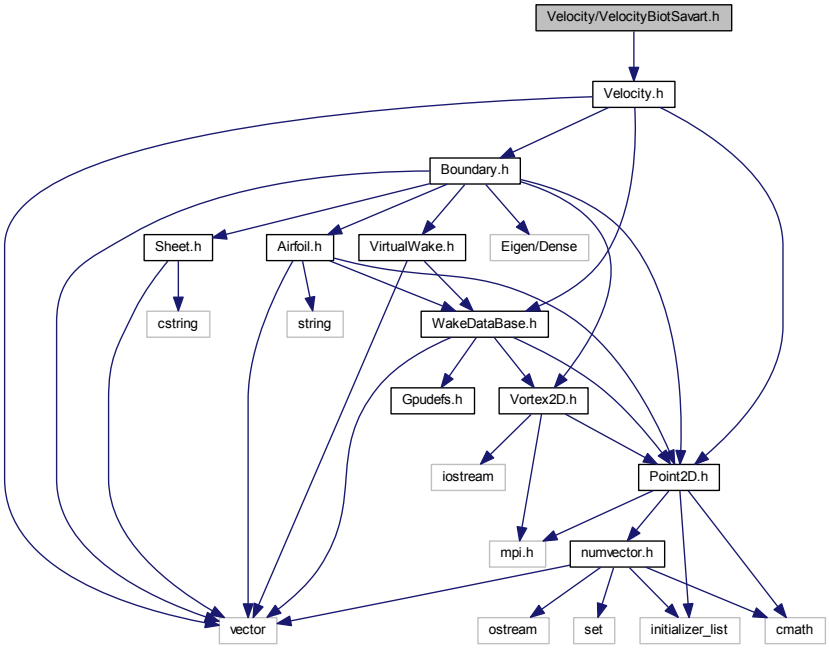
Граф вызова функции:



11.52 Файл Velocity/VelocityBiotSavart.h

Заголовочный файл с описанием класса VelocityBiotSavart.

#include "Velocity.h"
Граф включаемых заголовочных файлов для VelocityBiotSavart.h:



Граф файлов, в которые включается этот файл:

Классы

- class VelocityBiotSavart

  Класс, определяющий способ вычисления скоростей

11.52.1 Подробное описание

Заголовочный файл с описанием класса VelocityBiotSavart.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

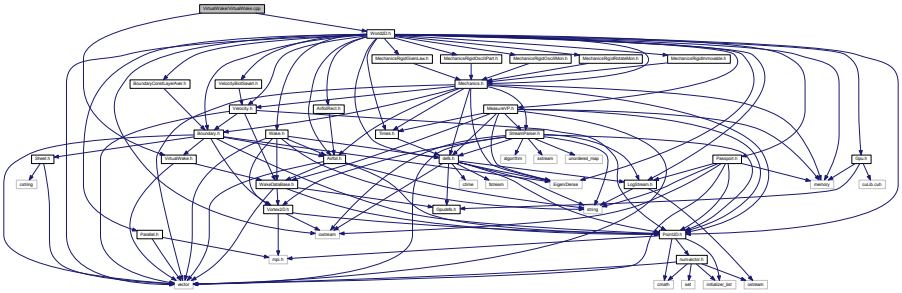
Дата

20 февраля 2019 г.

11.53 Файл VirtualWake/VirtualWake.cpp

Файл кода с описанием класса VirtualWake.

#include "VirtualWake.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для VirtualWake.cpp:



11.53.1 Подробное описание

Файл кода с описанием класса VirtualWake.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

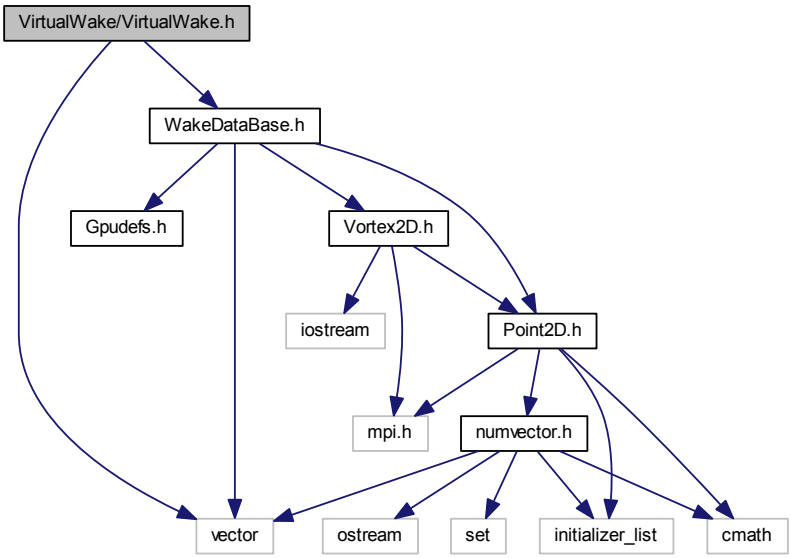
Дата

20 февраля 2019 г.

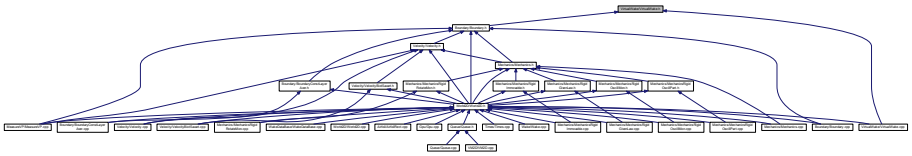
11.54 Файл VirtualWake/VirtualWake.h

Заголовочный файл с описанием класса VirtualWake.

#include <vector>
#include "WakeDataBase.h"
Граф включаемых заголовочных файлов для VirtualWake.h:



Граф файлов, в которые включается этот файл:



Классы

- class VirtualWake

  Класс, опеделяющий вихревой след (пелену)

11.54.1 Подробное описание

Заголовочный файл с описанием класса VirtualWake.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

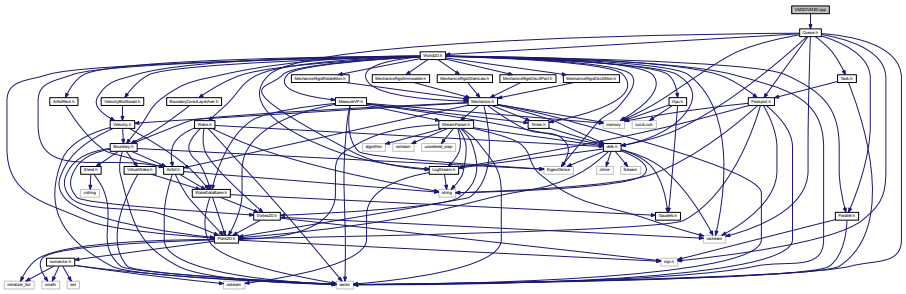
Дата

20 февраля 2019 г.

11.55 Файл VM2D/VM2D.cpp

Основной файл программы VM2D.

#include "Queue.h"
Граф включаемых заголовочных файлов для VM2D.cpp:



Функции

- void CreateMpiTypes ()
- void Initializers ()
- int main (int argc, char **argv)

11.55.1 Подробное описание

Основной файл программы VM2D.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

11.55.2 Функции

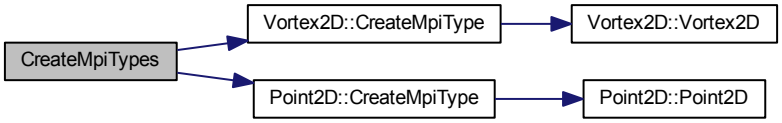
11.55.2.1 CreateMpiTypes()

void CreateMpiTypes ( )

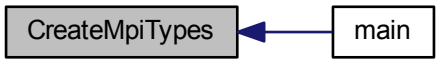
См. определение в файле VM2D.cpp строка 53

```
54 {
55     Vortex2D::CreateMpiType();
56     Point2D::CreateMpiType();
57 }
```

Граф вызовов:



Граф вызова функции:



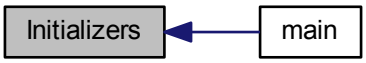
11.55.2.2 Initializers()

void Initializers ( )

См. определение в файле VM2D.cpp строка 59

```
60 {
61
62 }
```

Граф вызова функции:



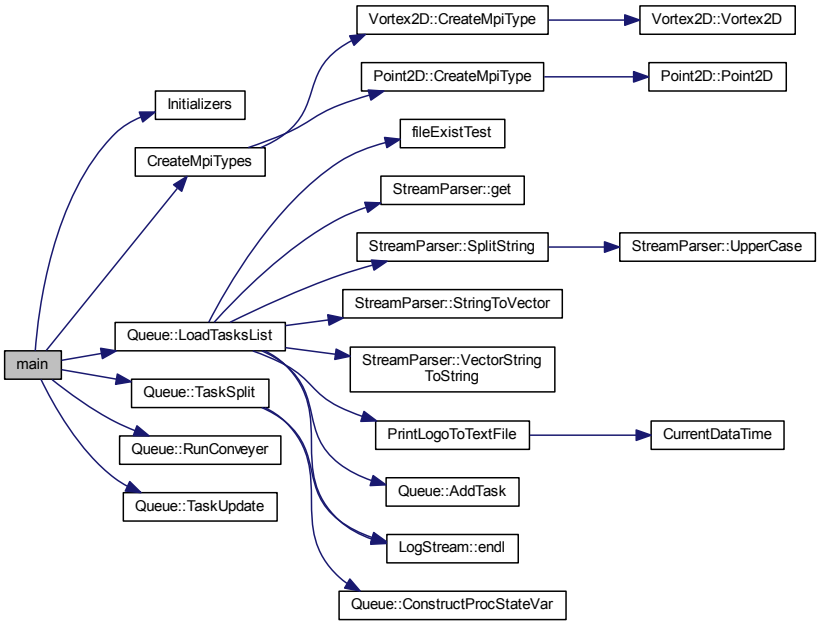
11.55.2.3 main()

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

См. определение в файле VM2D.cpp строка 65

```
{
    Initializers();

    Queue queue(argc, argv, CreateMpiTypes);

    queue.LoadTasksList("problems", "mechanics", "defaults", "switchers");

    do
    {
        queue.TaskSplit();

        queue.RunConveyer();

        queue.TaskUpdate();
    }
    while (queue.nextKvant);

    //cin.get();
}
```

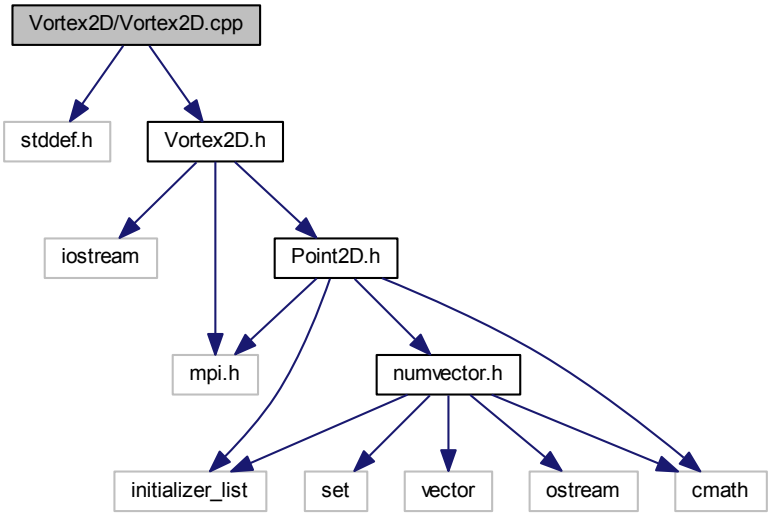
Граф вызовов:



11.56 Файл Vortex2D/Vortex2D.cpp

Файл кода с описанием класса Vortex2D.

#include <stddef.h>
#include "Vortex2D.h"
Граф включаемых заголовочных файлов для Vortex2D.cpp:



11.56.1 Подробное описание

Файл кода с описанием класса Vortex2D.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

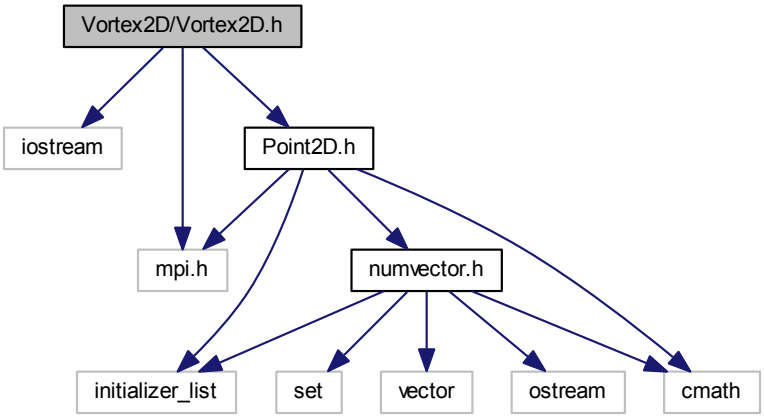
Дата

20 февраля 2019 г.

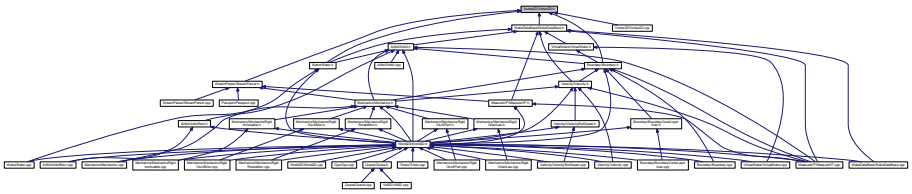
11.57 Файл Vortex2D/Vortex2D.h

Заголовочный файл с описанием класса Vortex2D.

#include <iostream>
#include "mpi.h"
#include "Point2D.h"
Граф включаемых заголовочных файлов для Vortex2D.h:



Граф файлов, в которые включается этот файл:



Классы

- class Vortex2D

  Класс, опеделяющий двумерный вихревой элемент

Определения типов

- typedef Vortex2D Source2D

11.57.1 Подробное описание

Заголовочный файл с описанием класса Vortex2D.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

11.57.2 Типы

11.57.2.1 Source2D

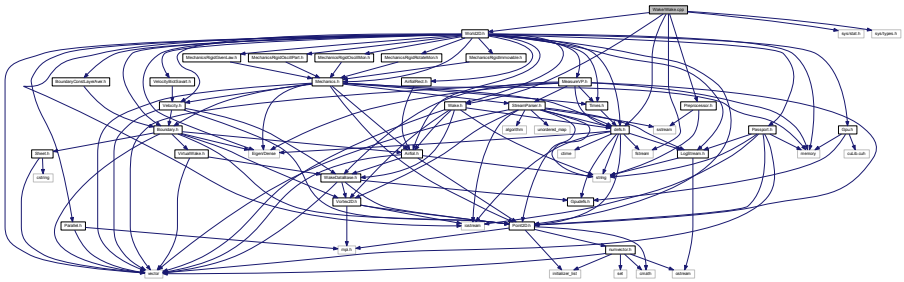
typedef Vortex2D Source2D

См. определение в файле Vortex2D.h строка 105

11.58 Файл Wake/Wake.cpp

Файл кода с описанием класса Wake.

#include <fstream>
#include <sys/stat.h>
#include <sys/types.h>
#include "Preprocessor.h"
#include "defs.h"
#include "Wake.h"
#include "World2D.h"
Граф включаемых заголовочных файлов для Wake.cpp:



11.58.1 Подробное описание

Файл кода с описанием класса Wake.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

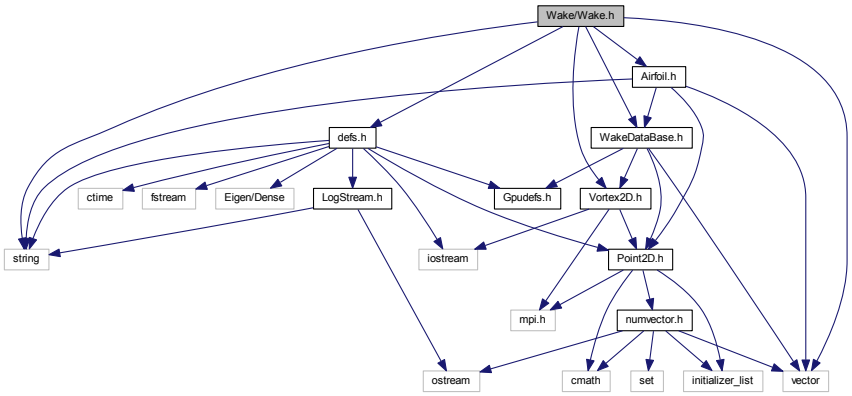
20 февраля 2019 г.

11.59 Файл Wake/Wake.h

Заголовочный файл с описанием класса Wake.

#include <string>
#include <vector>
#include "defs.h"
#include "WakeDataBase.h"
#include "Airfoil.h"

#include "Vortex2D.h"
Граф включаемых заголовочных файлов для Wake.h:



Граф файлов, в которые включается этот файл:

Классы

- class Wake

  Класс, опеделяющий вихревой след (пелену)

11.59.1 Подробное описание

Заголовочный файл с описанием класса Wake.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

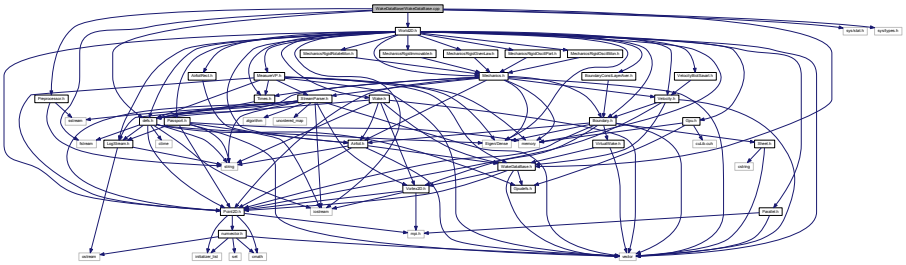
Дата

20 февраля 2019 г.

11.60 Файл WakeDataBase/WakeDataBase.cpp

Файл кода с описанием класса WakeDataBase.

#include <fstream>
#include <sys/stat.h>
#include <sys/types.h>
#include "Preprocessor.h"
#include "World2D.h"
#include "defs.h"
#include "WakeDataBase.h"
Граф включаемых заголовочных файлов для WakeDataBase.cpp:



11.60.1 Подробное описание

Файл кода с описанием класса WakeDataBase.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

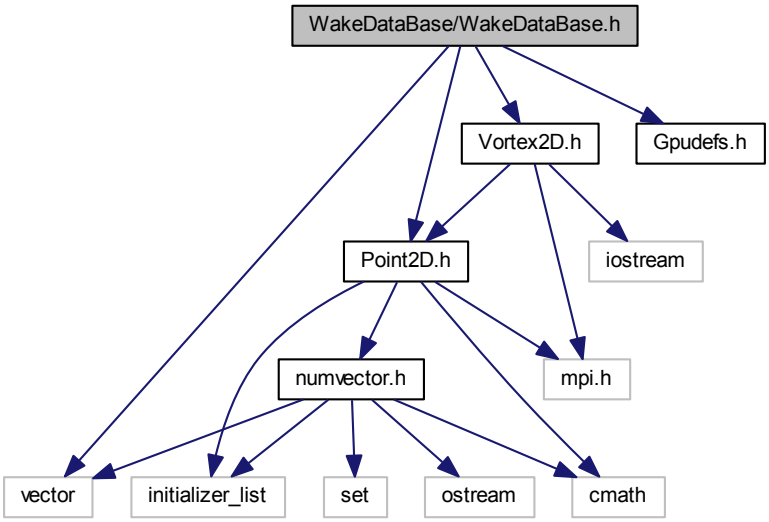
20 февраля 2019 г.

11.61 Файл WakeDataBase/WakeDataBase.h

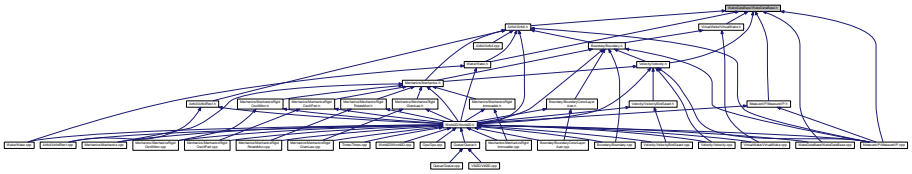
Заголовочный файл с описанием класса WakeDataBase.

#include <vector>
#include "Gpudefs.h"
#include "Point2D.h"

#include "Vortex2D.h"
Граф включаемых заголовочных файлов для WakeDataBase.h:



Граф файлов, в которые включается этот файл:



Классы

- class WakeDataBase

11.61.1 Подробное описание

Заголовочный файл с описанием класса WakeDataBase.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

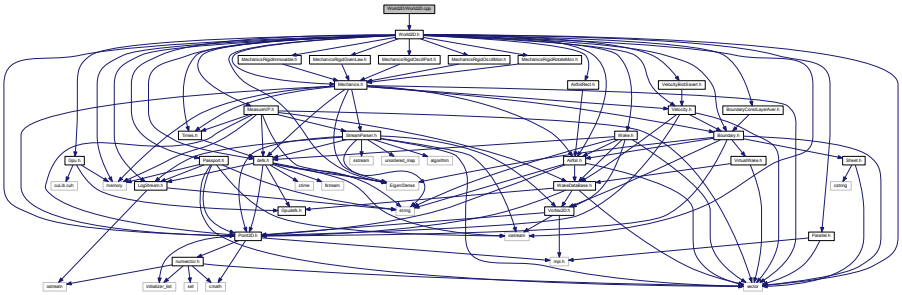
Дата

20 февраля 2019 г.

11.62 Файл World2D/World2D.cpp

Файл кода с описанием класса World2D.

#include "World2D.h"
Граф включаемых заголовочных файлов для World2D.cpp:



11.62.1 Подробное описание

Файл кода с описанием класса World2D.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

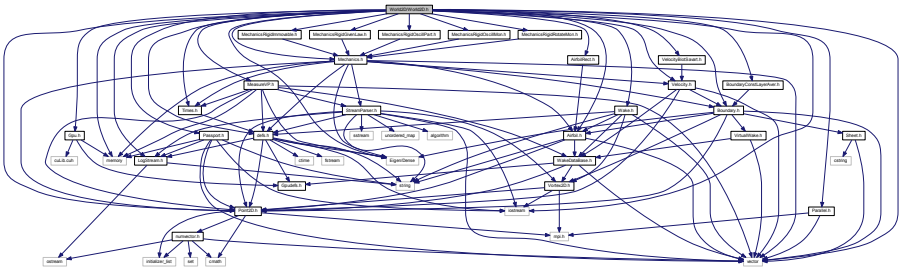
11.63 Файл World2D/World2D.h

Заголовочный файл с описанием класса World2D.

#include <iostream>
#include <memory>
#include <vector>
#include "Eigen/Dense"
#include "defs.h"
#include "Gpu.h"
#include "Airfoil.h"
#include "AirfoilRect.h"
#include "Boundary.h"
#include "BoundaryConstLayerAver.h"

#include "Mechanics.h"
#include "MechanicsRigidImmovable.h"
#include "MechanicsRigidGivenLaw.h"
#include "MechanicsRigidOscillPart.h"
#include "MechanicsRigidOscillMon.h"
#include "MechanicsRigidRotateMon.h"
#include "Velocity.h"
#include "VelocityBiotSavart.h"
#include "MeasureVP.h"
#include "LogStream.h"
#include "Parallel.h"
#include "Passport.h"
#include "Point2D.h"
#include "Times.h"
#include "Wake.h"

Граф включаемых заголовочных файлов для World2D.h:



Граф файлов, в которые включается этот файл:

Классы

- class World2D

  Класс, опеделяющий текущую решаемую задачу

11.63.1 Подробное описание

Заголовочный файл с описанием класса World2D.

Автор

Марчевский Илья Константинович
Кузьмина Ксения Сергеевна
Рятина Евгения Павловна

Версия

1.5

Дата

20 февраля 2019 г.

Предметный указатель

Параллельные функции, 17
CalcConvVeloToSetOfPoints, 18
CalcDiffVeloI1I2ToSetOfPoints, 18
GetConvVelocityToSetOfPoints, 19
GetConvVelocityToSetOfPointsFromVirtual↩ Vortexes, 20
GetDiffVelocityI0I3ToSetOfPointsAnd↩ ViscousStresses, 21
GetWakeInfluence, 22
Inside, 23
VirtualWakeSynchronize, 24
WakeSynchronize, 25
∼Airfoil
Airfoil, 38
∼AirfoilRect
AirfoilRect, 61
∼Boundary
Boundary, 84
∼BoundaryConstLayerAver
BoundaryConstLayerAver, 96
∼Gpu
Gpu, 122
∼LogStream
LogStream, 128
∼MeasureVP
MeasureVP, 134
∼Mechanics
Mechanics, 149
∼MechanicsRigidGivenLaw
MechanicsRigidGivenLaw, 168
∼MechanicsRigidImmovable
MechanicsRigidImmovable, 188
∼MechanicsRigidOscillMon
MechanicsRigidOscillMon, 207
∼MechanicsRigidOscillPart
MechanicsRigidOscillPart, 230
∼MechanicsRigidRotateMon
MechanicsRigidRotateMon, 253
∼Passport
Passport, 340
∼Point2D
Point2D, 358
∼Preprocessor
Preprocessor, 384
∼Queue
Queue, 397
∼Sheet
Sheet, 416
∼StreamParser
StreamParser, 424
∼Task
Task, 442
∼Times
Times, 448
∼Velocity
Velocity, 458
∼VelocityBiotSavart
VelocityBiotSavart, 465
∼VirtualWake
VirtualWake, 479
∼Vortex2D
Vortex2D, 484
∼Wake
Wake, 494
∼WakeDataBase
WakeDataBase, 513
∼World2D
World2D, 523

accelCft
PhysicalProperties, 348
addCurrTime
PhysicalProperties, 349
AddTask
Queue, 397
additionalWake
MeasureVP, 143
afl
Boundary, 91
BoundaryConstLayerAver, 118
Mechanics, 160
MechanicsRigidGivenLaw, 180
MechanicsRigidImmovable, 199
MechanicsRigidOscillMon, 221
MechanicsRigidOscillPart, 242
MechanicsRigidRotateMon, 270
aflPan
VirtualWake, 480
afterSlashParser
Preprocessor, 384
Airfoil, 35
∼Airfoil, 38
Airfoil, 38
CalcNrmTauLen, 39
gammaThrough, 50
GetGabarits, 40
IFCUDA, 41, 42
inverse, 50

isAfter, 42
isInsideGabarits, 43
isOutsideGabarits, 44
IsPointInAirfoil, 44
J, 50
len, 51
lowLeft, 51
m, 51
Move, 46
np, 51
nrm, 51
numberInPassport, 52
phiAfl, 52
r, 52
rcm, 52
ReadFromFile, 47
Rotate, 48
Scale, 49
tau, 52
upRight, 53
v, 53
viscousStress, 53
W, 53
airfoil
World2D, 555
Airfoil/Airfoil.cpp, 561
Airfoil/Airfoil.h, 562
Airfoil/AirfoilRect.cpp, 563
Airfoil/AirfoilRect.h, 564
AirfoilParams, 54
angle, 55
basePoint, 55
boundaryCondition, 56
fileAirfoil, 56
inverse, 56
mechanicalSystem, 56
mechanicalSystemParameters, 56
mechanicalSystemType, 57
panelsType, 57
scale, 57
airfoilParams
Passport, 344
AirfoilRect, 58
~AirfoilRect, 61
AirfoilRect, 60, 61
CalcNrmTauLen, 62
gammaThrough, 77
GetDiffVelocityI0I3ToSetOfPointsAnd↩ ViscousStresses, 63
GetGabarits, 66
IFCUDA, 67, 68
inverse, 77
isAfter, 68
isInsideGabarits, 69
isOutsideGabarits, 70
IsPointInAirfoil, 70
J, 77
len, 77
lowLeft, 78
m, 78
Move, 72
np, 78
nrm, 78
numberInPassport, 78
phiAfl, 79
r, 79
rcm, 79
ReadFromFile, 73
Rotate, 75
Scale, 76
tau, 79
upRight, 79
v, 80
viscousStress, 80
W, 80
airfoilsDir
Passport, 345
Alpha
BoundaryConstLayerAver, 97
MechanicsRigidRotateMon, 253
angle
AirfoilParams, 55
assignStream
LogStream, 128
attachedSourceSheet
Sheet, 417
attachedVortexSheet
Sheet, 418

b
MechanicsRigidOscillMon, 221
MechanicsRigidOscillPart, 243
MechanicsRigidRotateMon, 270
basePoint
AirfoilParams, 55
bnd
VirtualWake, 480
boundDenom
defs.h, 581
Boundary, 81
~Boundary, 84
afl, 91
Boundary, 83
CC, 91
ComputeAttachedSheetsIntensity, 85
FillMatrixFromOther, 86
FillMatrixSelf, 86
FillRhs, 87
FillRhsFromOther, 88
GetConvVelocityAtVirtualVortexes, 89
GetUnknownsSize, 89
numberInPassport, 91
oldSheets, 91
sheetDim, 92
sheets, 92
SolutionToFreeVortexSheetAndVirtualVortex, 90

virtualWake, 92
vortexPerPanel, 92
W, 93
boundary
Mechanics, 160
MechanicsRigidGivenLaw, 180
MechanicsRigidImmovable, 199
MechanicsRigidOscillMon, 221
MechanicsRigidOscillPart, 243
MechanicsRigidRotateMon, 270
World2D, 555
Boundary/Boundary.cpp, 565
Boundary/Boundary.h, 566
Boundary/BoundaryConstLayerAver.cpp, 567
Boundary/BoundaryConstLayerAver.h, 568
boundaryCondition
AirfoilParams, 56
BoundaryConstLayerAver, 93
~BoundaryConstLayerAver, 96
afl, 118
Alpha, 97
BoundaryConstLayerAver, 95
CC, 118
ComputeAttachedSheetsIntensity, 98
FillMatrixFromOther, 98
FillMatrixSelf, 100
FillRhs, 102
FillRhsFromOther, 104
GetConvVelocityAtVirtualVortexes, 106
GetConvVelocityToSetOfPoints, 107
GetConvVelocityToSetOfPointsFromVirtual↩ Vortexes, 109
GetUnknownsSize, 112
GetWakeInfluence, 112
Lambda, 114
numberInPassport, 118
oldSheets, 118
Omega, 115
sheetDim, 119
sheets, 119
SolutionToFreeVortexSheetAndVirtualVortex, 116
virtualWake, 119
vortexPerPanel, 119
W, 120

CU_CONV_TOBOU
Gpudefs.h, 595
CU_CONV_TOWAKE
Gpudefs.h, 595
CU_CONVVIRT
Gpudefs.h, 595
CU_I0I3
Gpudefs.h, 595
CU_I1I2
Gpudefs.h, 596
CU_PAIRS
Gpudefs.h, 596
CU_RHS
Gpudefs.h, 596
CUBLOCK
Gpudefs.h, 596
CUDA_DISABLE
Gpudefs.h, 596
CUDA_ENABLE
Gpudefs.h, 596
CUDAdef
Gpudefs.h, 597
CalcConvVelo
Velocity, 458
VelocityBiotSavart, 465
CalcConvVeloToSetOfPoints
Параллельные функции, 18
VelocityBiotSavart, 466
CalcDiffVelo
Velocity, 459
VelocityBiotSavart, 469
CalcDiffVeloI1I2ToSetOfPoints
Параллельные функции, 18
VelocityBiotSavart, 471
CalcDiffVeloI1I2ToSetOfPointsFromPanels
Velocity, 461
VelocityBiotSavart, 473
CalcNrmTauLen
Airfoil, 39
AirfoilRect, 62
CalcSaveVP
MeasureVP, 135
CalcVortexVelo
World2D, 524
CC
Boundary, 91
BoundaryConstLayerAver, 118
CheckInside
World2D, 530
Collaps
Wake, 495
collapseRightBorderParameter
Wake, 510
collapseScaleParameter
Wake, 510
commSolving
Queue, 408
commStarting
Queue, 408
commWork
Parallel, 333
ComputeAttachedSheetsIntensity
Boundary, 85
BoundaryConstLayerAver, 98
ComputeDisplacementRcm
MechanicsRigidImmovable, 189
ConstructProcStateVar
Queue, 398
convVelo
VortexesParams, 490
CreateMpiType

Point2D, 358
Vortex2D, 484
CreateMpiTypes
VM2D.cpp, 662
cross
numvector.h, 619
cross3
numvector.h, 620
cuda
World2D, 556
currTime
PhysicalProperties, 351
CurrentDataTime
defs.cpp, 570
defs.h, 581
currentKvant
Queue, 408
currentParser
Preprocessor, 391
currentStep
World2D, 556

DEPRECATED
numvector.h, 618
DPI
defs.h, 591
DYN_SCHEDULE
defs.h, 581
database
StreamParser, 439
defaultAirfoil
defaults, 28
defaultAirfoilsDir
defaults, 30
defaultAngle
defaults, 30
defaultBasePoint
defaults, 30
defaultBoundaryCondition
defaults, 30
defaultCopyPspFile
defaults, 31
defaultDelta
defaults, 31
defaultDistFar
defaults, 31
defaultFileAirfoil
defaults, 29
defaultFileSource
defaults, 29
defaultFileWake
defaults, 29
defaultInverse
defaults, 31
defaultMechanicalSystem
defaults, 31
defaultMechanicalSystemType
defaults, 32
defaultNameLength
defaults, 32
defaultNp
defaults, 32
defaultPanelsType
defaults, 32
defaultPspFile
defaults, 32
defaultQueueLogStream
defaults, 33
defaultSaveTXT
defaults, 33
defaultSaveVTK
defaults, 33
defaultSaveVP
defaults, 33
defaultScale
defaults, 33
defaultTimeAccel
defaults, 34
defaultTimeStart
defaults, 34
defaultVRef
defaults, 34
defaultVortexPerPanel
defaults, 34
defaultWakesDir
defaults, 34
defaultWorld2DLogStream
defaults, 34
defaults, 27
defaultAirfoil, 28
defaultAirfoilsDir, 30
defaultAngle, 30
defaultBasePoint, 30
defaultBoundaryCondition, 30
defaultCopyPspFile, 31
defaultDelta, 31
defaultDistFar, 31
defaultFileAirfoil, 29
defaultFileSource, 29
defaultFileWake, 29
defaultInverse, 31
defaultMechanicalSystem, 31
defaultMechanicalSystemType, 32
defaultNameLength, 32
defaultNp, 32
defaultPanelsType, 32
defaultPspFile, 32
defaultQueueLogStream, 33
defaultSaveTXT, 33
defaultSaveVTK, 33
defaultSaveVP, 33
defaultScale, 33
defaultTimeAccel, 34
defaultTimeStart, 34
defaultVRef, 34
defaultVortexPerPanel, 34
defaultWakesDir, 34

defaultWorld2DLogStream, 34
StreamParser, 439
defs.cpp
CurrentDataTime, 570
M4, 571
Months, 577
PrintHeaderToTextFile, 572
PrintLogoToTextFile, 573
SaveToStream, 575, 576
defs.h
boundDenom, 581
CurrentDataTime, 581
DPI, 591
DYN_SCHEDULE, 581
fileExistTest, 582
fileNameStep, 583
IDPI, 592
M4, 584
operator<<, 585
PI, 592
PrintHeaderToTextFile, 586
PrintLogoToTextFile, 587
SaveToStream, 588–590
sqr, 590
timePeriod, 581
defs/defs.cpp, 569
defs/defs.h, 577
degOfFreedom
Mechanics, 160
MechanicsRigidGivenLaw, 180
MechanicsRigidImmovable, 200
MechanicsRigidOscillMon, 221
MechanicsRigidOscillPart, 243
MechanicsRigidRotateMon, 271
delta
WakeDiscretizationProperties, 516
diffVelo
VortexesParams, 490
dir
Passport, 345
disp
parProp, 335
dist
numvector.h, 621
dist2
numvector.h, 622
dist2To
numvector, 307
Point2D, 359
distFar
WakeDiscretizationProperties, 516
distTo
numvector, 308
Point2D, 360
domainRadius
MeasureVP, 143
dot
numvector.h, 623, 624
dT
Times, 449
dt
TimeDiscretizationProperties, 444

endl
LogStream, 129
eps
WakeDiscretizationProperties, 517
eps2
WakeDiscretizationProperties, 517
epsastWake
VortexesParams, 490
epscol
WakeDiscretizationProperties, 517

fileAirfoil
AirfoilParams, 56
fileExistTest
defs.h, 582
fileNameStep
defs.h, 583
fileSource
WakeDiscretizationProperties, 517
fileWake
WakeDiscretizationProperties, 517
FillAtt
Mechanics, 150
MechanicsRigidGivenLaw, 169
MechanicsRigidImmovable, 189
MechanicsRigidOscillMon, 208
MechanicsRigidOscillPart, 230
MechanicsRigidRotateMon, 254
FillMatrixAndRhs
World2D, 531
FillMatrixFromOther
Boundary, 86
BoundaryConstLayerAver, 98
FillMatrixSelf
Boundary, 86
BoundaryConstLayerAver, 100
FillMechanicsRhs
Mechanics, 150
MechanicsRigidGivenLaw, 169
MechanicsRigidImmovable, 189
MechanicsRigidOscillMon, 209
MechanicsRigidOscillPart, 231
MechanicsRigidRotateMon, 256
FillMechanicsRowsAndCross
Mechanics, 151
MechanicsRigidGivenLaw, 170
MechanicsRigidImmovable, 190
MechanicsRigidOscillMon, 210
MechanicsRigidOscillPart, 232
MechanicsRigidRotateMon, 257
FillRhs
Boundary, 87
BoundaryConstLayerAver, 102
FillRhsFromOther

Boundary, 88
BoundaryConstLayerAver, 104
finished
Queue, 409
flagFinish
Queue, 409
freeVortexSheet
Sheet, 418

g
Vortex2D, 485, 486
gam
Vortex2D, 487
gammaThrough
Airfoil, 50
AirfoilRect, 77
GenerateForcesHeader
Mechanics, 151
MechanicsRigidGivenLaw, 170
MechanicsRigidImmovable, 190
MechanicsRigidOscillMon, 211
MechanicsRigidOscillPart, 232
MechanicsRigidRotateMon, 259
GenerateForcesString
Mechanics, 152
MechanicsRigidGivenLaw, 171
MechanicsRigidImmovable, 191
MechanicsRigidOscillMon, 212
MechanicsRigidOscillPart, 233
MechanicsRigidRotateMon, 260
GenerateMechanicsHeader
World2D, 534
GeneratePositionHeader
Mechanics, 153
MechanicsRigidGivenLaw, 172
MechanicsRigidImmovable, 192
MechanicsRigidOscillMon, 213
MechanicsRigidOscillPart, 234
MechanicsRigidRotateMon, 261
GeneratePositionString
Mechanics, 154
MechanicsRigidGivenLaw, 173
MechanicsRigidImmovable, 193
MechanicsRigidOscillMon, 214
MechanicsRigidOscillPart, 235
MechanicsRigidRotateMon, 262
GenerateStatHeader
Times, 449
GenerateStatString
Times, 450
get
StreamParser, 424–429
getAirfoil
World2D, 534
GetAllParamsFromParser
Passport, 340
getBoundary
World2D, 535
GetConvVelocityAtVirtualVortexes
Boundary, 89
BoundaryConstLayerAver, 106
GetConvVelocityToSetOfPoints
Параллельные функции, 19
BoundaryConstLayerAver, 107
GetConvVelocityToSetOfPointsFromVirtual↩
Vortexes
Параллельные функции, 20
BoundaryConstLayerAver, 109
getCuda
World2D, 536
getCurrTime
PhysicalProperties, 349
getCurrentStep
World2D, 536
GetDiffVelocityI0I3ToSetOfPointsAndViscous↩
Stresses
Параллельные функции, 21
AirfoilRect, 63
GetGabarits
Airfoil, 40
AirfoilRect, 66
GetHydroDynamForce
Mechanics, 155
MechanicsRigidGivenLaw, 174
MechanicsRigidImmovable, 194
MechanicsRigidOscillMon, 215
MechanicsRigidOscillPart, 236
MechanicsRigidRotateMon, 263
getInfo
World2D, 537, 538
GetMz
MechanicsRigidRotateMon, 264
getNonConstAirfoil
World2D, 539
getNonConstBoundary
World2D, 539
getNonConstVelocity
World2D, 540
getNumberOfAirfoil
World2D, 540
getNumberOfBoundary
World2D, 541
GetPairs
Wake, 497
getParallel
World2D, 541
getPassport
World2D, 542
getSource
World2D, 543
getTimestat
World2D, 544
GetUnknownsSize
Boundary, 89
BoundaryConstLayerAver, 112
getVelocity
World2D, 545

getWake
World2D, 545
GetWakeInfluence
Параллельные функции, 22
BoundaryConstLayerAver, 112
Gpu, 120
~Gpu, 122
Gpu, 121
setAccelCoeff, 124
setCollapseCoeff, 125
W, 125
Gpu/Gpu.cpp, 592
Gpu/Gpu.h, 593
Gpu/Gpudefs.h, 594
Gpudefs.h
CU_CONV_TOBOU, 595
CU_CONV_TOWAKE, 595
CU_CONVVIRT, 595
CU_I0I3, 595
CU_I1I2, 596
CU_PAIRS, 596
CU_RHS, 596
CUBLOCK, 596
CUDA_DISABLE, 596
CUDA_ENABLE, 596
CUDAdef, 597
IFCUDA_, 597
IFCUDA, 597
INC_VORT_DEV, 597
groupAll
Queue, 409
groupSolving
Queue, 409
groupStarting
Queue, 410

historyPoints
MeasureVP, 143
hydroDynamForce
Mechanics, 160
MechanicsRigidGivenLaw, 180
MechanicsRigidImmovable, 200
MechanicsRigidOscillMon, 222
MechanicsRigidOscillPart, 243
MechanicsRigidRotateMon, 271
hydroDynamMoment
Mechanics, 161
MechanicsRigidGivenLaw, 181
MechanicsRigidImmovable, 200
MechanicsRigidOscillMon, 222
MechanicsRigidOscillPart, 243
MechanicsRigidRotateMon, 271

I0
VortexesParams, 490
I1
VortexesParams, 491
I2
VortexesParams, 491
I3
VortexesParams, 491
IDPI
defs.h, 592
IFCUDA_
Gpudefs.h, 597
IFCUDA
Airfoil, 41, 42
AirfoilRect, 67, 68
Gpudefs.h, 597
INC_VORT_DEV
Gpudefs.h, 597
inCharLastEscape
Preprocessor, 392
inCharParser
Preprocessor, 385
inInlineCommentParser
Preprocessor, 386
inInlineLastEscape
Preprocessor, 392
inMultilineCommentParser
Preprocessor, 387
inMultilineLastStar
Preprocessor, 392
inStringLastEscape
Preprocessor, 392
inStringParser
Preprocessor, 388
info
Passport, 345
Queue, 410
StreamParser, 439
World2D, 556
inheritStream
LogStream, 129
initialInput
Preprocessor, 392
initialPoints
MeasureVP, 144
Initializers
VM2D.cpp, 662
Inside
Параллельные функции, 23
intermediateOutput
Preprocessor, 393
invMatr
World2D, 556
inverse
Airfoil, 50
AirfoilParams, 56
AirfoilRect, 77
isAfter
Airfoil, 42
AirfoilRect, 68
isDeform
Mechanics, 161
MechanicsRigidGivenLaw, 181
MechanicsRigidImmovable, 200

MechanicsRigidOscillMon, 222
MechanicsRigidOscillPart, 244
MechanicsRigidRotateMon, 271
isFinished
World2D, 546
isInsideGabarits
Airfoil, 43
AirfoilRect, 69
isMoves
Mechanics, 161
MechanicsRigidGivenLaw, 181
MechanicsRigidImmovable, 200
MechanicsRigidOscillMon, 222
MechanicsRigidOscillPart, 244
MechanicsRigidRotateMon, 271
isOutsideGabarits
Airfoil, 44
AirfoilRect, 70
IsPointInAirfoil
Airfoil, 44
AirfoilRect, 70
isRotate
Mechanics, 161
MechanicsRigidGivenLaw, 181
MechanicsRigidImmovable, 201
MechanicsRigidOscillMon, 222
MechanicsRigidOscillPart, 244
MechanicsRigidRotateMon, 272

J
Airfoil, 50
AirfoilRect, 77
MechanicsRigidRotateMon, 272

k
MechanicsRigidOscillMon, 223
MechanicsRigidOscillPart, 244
MechanicsRigidRotateMon, 272
kcross
numvector, 309
Point2D, 360
KronProd
numvector.h, 624
kvantTime
Queue, 410

label
LogStream, 131
Lambda
BoundaryConstLayerAver, 114
MechanicsRigidRotateMon, 265
len
Airfoil, 51
AirfoilRect, 77
parProp, 335
length
numvector, 310
Point2D, 361
length2
numvector, 310
Point2D, 361
linearSystemSolver
NumericalSchemes, 277
LoadTasksList
Queue, 399
LogStream, 126
~LogStream, 128
assignStream, 128
endl, 129
inheritStream, 129
label, 131
LogStream, 127
operator(), 130
pStr, 131
LogStream/LogStream.cpp, 598
LogStream/LogStream.h, 599
lowLeft
Airfoil, 51
AirfoilRect, 78

m
Airfoil, 51
AirfoilRect, 78
MechanicsRigidOscillMon, 223
MechanicsRigidOscillPart, 244
M4
defs.cpp, 571
defs.h, 584
main
VM2D.cpp, 663
matDotVec
numvector.h, 625
matr
World2D, 556
MeasureVP/MeasureVP.cpp, 600
MeasureVP/MeasureVP.h, 601
MeasureVP, 132
~MeasureVP, 134
additionalWake, 143
CalcSaveVP, 135
domainRadius, 143
historyPoints, 143
initialPoints, 144
MeasureVP, 133
pressure, 144
ReadPointsFromFile, 142
velocity, 144
W, 144
measureVP
World2D, 557
mechParamsParser
Mechanics, 161
MechanicsRigidGivenLaw, 181
MechanicsRigidImmovable, 201
MechanicsRigidOscillMon, 223
MechanicsRigidOscillPart, 245
MechanicsRigidRotateMon, 272
mechanicalSystem

AirfoilParams, 56
mechanicalSystemParameters
AirfoilParams, 56
mechanicalSystemType
AirfoilParams, 57
Mechanics, 145
~Mechanics, 149
afl, 160
boundary, 160
degOfFreedom, 160
FillAtt, 150
FillMechanicsRhs, 150
FillMechanicsRowsAndCross, 151
GenerateForcesHeader, 151
GenerateForcesString, 152
GeneratePositionHeader, 153
GeneratePositionString, 154
GetHydroDynamForce, 155
hydroDynamForce, 160
hydroDynamMoment, 161
isDeform, 161
isMoves, 161
isRotate, 161
mechParamsParser, 161
Mechanics, 148
Move, 156
numberInPassport, 162
Phi, 162
Phi0, 162
PhiOld, 162
PositionOfAirfoilRcm, 156
Rcm, 162
Rcm0, 162
RcmOld, 163
ReadParametersFromDictionary, 157
ReadSpecificParametersFromDictionary, 158
SolutionToMechanicalSystem, 158
Vcm, 163
Vcm0, 163
VcmOld, 163
VeloOfAirfoilPanels, 158
VeloOfAirfoilRcm, 159
virtVortParams, 163
W, 164
Wcm, 164
Wcm0, 164
WcmOld, 164
mechanics
World2D, 557
Mechanics/Mechanics.cpp, 602
Mechanics/Mechanics.h, 602
Mechanics/MechanicsRigidGivenLaw.cpp, 604
Mechanics/MechanicsRigidGivenLaw.h, 604
Mechanics/MechanicsRigidImmovable.cpp, 606
Mechanics/MechanicsRigidImmovable.h, 606
Mechanics/MechanicsRigidOscillMon.cpp, 607
Mechanics/MechanicsRigidOscillMon.h, 608
Mechanics/MechanicsRigidOscillPart.cpp, 609
Mechanics/MechanicsRigidOscillPart.h, 610
Mechanics/MechanicsRigidRotateMon.cpp, 611
Mechanics/MechanicsRigidRotateMon.h, 611
MechanicsRigidGivenLaw, 165
~MechanicsRigidGivenLaw, 168
afl, 180
boundary, 180
degOfFreedom, 180
FillAtt, 169
FillMechanicsRhs, 169
FillMechanicsRowsAndCross, 170
GenerateForcesHeader, 170
GenerateForcesString, 171
GeneratePositionHeader, 172
GeneratePositionString, 173
GetHydroDynamForce, 174
hydroDynamForce, 180
hydroDynamMoment, 181
isDeform, 181
isMoves, 181
isRotate, 181
mechParamsParser, 181
MechanicsRigidGivenLaw, 167
Move, 176
numberInPassport, 182
Phi, 182
Phi0, 182
PhiOld, 182
PositionOfAirfoilRcm, 176
Rcm, 182
Rcm0, 182
RcmOld, 183
ReadSpecificParametersFromDictionary, 177
SolutionToMechanicalSystem, 177
Vcm, 183
Vcm0, 183
VcmOld, 183
VeloOfAirfoilPanels, 178
VeloOfAirfoilRcm, 179
virtVortParams, 183
W, 184
Wcm, 184
Wcm0, 184
WcmOld, 184
MechanicsRigidImmovable, 185
~MechanicsRigidImmovable, 188
afl, 199
boundary, 199
ComputeDisplacementRcm, 189
degOfFreedom, 200
FillAtt, 189
FillMechanicsRhs, 189
FillMechanicsRowsAndCross, 190
GenerateForcesHeader, 190
GenerateForcesString, 191
GeneratePositionHeader, 192
GeneratePositionString, 193
GetHydroDynamForce, 194

hydroDynamForce, 200
hydroDynamMoment, 200
isDeform, 200
isMoves, 200
isRotate, 201
mechParamsParser, 201
MechanicsRigidImmovable, 187
Move, 196
numberInPassport, 201
Phi, 201
Phi0, 201
PhiOld, 202
PositionOfAirfoilRcm, 196
Rcm, 202
Rcm0, 202
RcmOld, 202
ReadSpecificParametersFromDictionary, 197
SolutionToMechanicalSystem, 197
Vcm, 202
Vcm0, 203
VcmOld, 203
VeloOfAirfoilPanels, 198
VeloOfAirfoilRcm, 198
virtVortParams, 203
W, 203
Wcm, 203
Wcm0, 204
WcmOld, 204
MechanicsRigidOscillMon, 204
~MechanicsRigidOscillMon, 207
afl, 221
b, 221
boundary, 221
degOfFreedom, 221
FillAtt, 208
FillMechanicsRhs, 209
FillMechanicsRowsAndCross, 210
GenerateForcesHeader, 211
GenerateForcesString, 212
GeneratePositionHeader, 213
GeneratePositionString, 214
GetHydroDynamForce, 215
hydroDynamForce, 222
hydroDynamMoment, 222
isDeform, 222
isMoves, 222
isRotate, 222
k, 223
m, 223
mechParamsParser, 223
MechanicsRigidOscillMon, 207
Move, 216
numberInPassport, 223
Phi, 223
Phi0, 223
PhiOld, 224
PositionOfAirfoilRcm, 217
Rcm, 224
Rcm0, 224
RcmOld, 224
ReadSpecificParametersFromDictionary, 217
SolutionToMechanicalSystem, 218
Vcm, 224
Vcm0, 225
VcmOld, 225
VeloOfAirfoilPanels, 219
VeloOfAirfoilRcm, 220
virtVortParams, 225
W, 225
Wcm, 225
Wcm0, 226
WcmOld, 226
MechanicsRigidOscillPart, 226
~MechanicsRigidOscillPart, 230
afl, 242
b, 243
boundary, 243
degOfFreedom, 243
FillAtt, 230
FillMechanicsRhs, 231
FillMechanicsRowsAndCross, 232
GenerateForcesHeader, 232
GenerateForcesString, 233
GeneratePositionHeader, 234
GeneratePositionString, 235
GetHydroDynamForce, 236
hydroDynamForce, 243
hydroDynamMoment, 243
isDeform, 244
isMoves, 244
isRotate, 244
k, 244
m, 244
mechParamsParser, 245
MechanicsRigidOscillPart, 229
Move, 237
numberInPassport, 245
Phi, 245
Phi0, 245
PhiOld, 245
PositionOfAirfoilRcm, 238
Rcm, 245
Rcm0, 246
RcmOld, 246
ReadSpecificParametersFromDictionary, 239
SolutionToMechanicalSystem, 240
u, 246
u0, 246
uOld, 246
Vcm, 247
Vcm0, 247
VcmOld, 247
VeloOfAirfoilPanels, 241
VeloOfAirfoilRcm, 242
virtVortParams, 247
W, 247

Wcm, 248
Wcm0, 248
WcmOld, 248
y, 248
y0, 248
yOld, 248
MechanicsRigidRotateMon, 249
~MechanicsRigidRotateMon, 253
afl, 270
Alpha, 253
b, 270
boundary, 270
degOfFreedom, 271
FillAtt, 254
FillMechanicsRhs, 256
FillMechanicsRowsAndCross, 257
GenerateForcesHeader, 259
GenerateForcesString, 260
GeneratePositionHeader, 261
GeneratePositionString, 262
GetHydroDynamForce, 263
GetMz, 264
hydroDynamForce, 271
hydroDynamMoment, 271
isDeform, 271
isMoves, 271
isRotate, 272
J, 272
k, 272
Lambda, 265
mechParamsParser, 272
MechanicsRigidRotateMon, 252
Move, 265
Mz, 272
numberInPassport, 273
Phi, 273
Phi0, 273
PhiOld, 273
PositionOfAirfoilRcm, 266
Rcm, 273
Rcm0, 273
RcmOld, 274
ReadSpecificParametersFromDictionary, 267
SolutionToMechanicalSystem, 268
tMomentAccel, 274
tRotateAccel, 274
Vcm, 274
Vcm0, 274
VcmOld, 275
VeloOfAirfoilPanels, 269
VeloOfAirfoilRcm, 269
virtVortParams, 275
W, 275
Wcm, 275
Wcm0, 275
WcmOld, 276
member
numvector, 311
Point2D, 362
ModifyE2
VelocityBiotSavart.cpp, 656
Months
defs.cpp, 577
Move
Airfoil, 46
AirfoilRect, 72
Mechanics, 156
MechanicsRigidGivenLaw, 176
MechanicsRigidImmovable, 196
MechanicsRigidOscillMon, 216
MechanicsRigidOscillPart, 237
MechanicsRigidRotateMon, 265
MoveInside
Wake, 499
MoveInsideMovingBoundary
Wake, 501
MoveVortexes
World2D, 547
mpiPoint2D
Point2D, 379
mpiVortex2D
Vortex2D, 487
myDisp
parProp, 336
myLen
parProp, 336
myProcState
Queue, 411
myProcStateVar
Queue, 411
myidAll
Queue, 410
myidWork
Parallel, 333
Mz
MechanicsRigidRotateMon, 272

nProc
Task, 442
nProcAll
Queue, 412
nProcWork
Parallel, 334
nameLength
TimeDiscretizationProperties, 444
neighb
Wake, 510
nextKvant
Queue, 411
norm1
nummatrix, 285
numvector, 312
Point2D, 362
normalParser
Preprocessor, 389
normalize
numvector, 312

Point2D, 363
norminf
nummatrix, 286
numvector, 313
Point2D, 363
np
Airfoil, 51
AirfoilRect, 78
nrm
Airfoil, 51
AirfoilRect, 78
nu
PhysicalProperties, 351
numberInPassport
Airfoil, 52
AirfoilRect, 78
Boundary, 91
BoundaryConstLayerAver, 118
Mechanics, 162
MechanicsRigidGivenLaw, 182
MechanicsRigidImmovable, 201
MechanicsRigidOscillMon, 223
MechanicsRigidOscillPart, 245
MechanicsRigidRotateMon, 273
numberOfTask
Queue, 412
NumericalSchemes, 276
linearSystemSolver, 277
velocityComputation, 277
wakeMotionIntegrator, 277
numericalSchemes
Passport, 345
nummatrix
norm1, 285
norminf, 286
nummatrix, 280, 281, 283–285
operator &, 286, 287
operator std::vector< std::vector< T >>, 288
operator!=, 288
operator*, 289
operator*=, 289
operator+, 290
operator+=, 291
operator-, 291, 292
operator-=, 293
operator/=, 293
operator==, 295
operator[], 296
r, 299
size, 297
toIdentity, 297
toZero, 298
transpose, 298
nummatrix< T, n, m >, 277
nummatrix.h
operator &, 613
operator<<, 614
operator|, 615
numvector
dist2To, 307
distTo, 308
kcross, 309
length, 310
length2, 310
member, 311
norm1, 312
normalize, 312
norminf, 313
numvector, 303–307
operator &, 313
operator std::set< P >, 314
operator std::vector< P >, 315
operator!=, 315
operator*, 316
operator*=, 316
operator^, 324, 325
operator+, 317, 318
operator+=, 318
operator-, 319, 320
operator-=, 320
operator/=, 321
operator=, 321
operator==, 322
operator[], 322, 324
r, 330
rotateLeft, 326
size, 326
toSet, 327
toVector, 327
toZero, 327
unit, 328
numvector< T, n >, 299
numvector.h
cross, 619
cross3, 620
DEPRECATED, 618
dist, 621
dist2, 622
dot, 623, 624
KronProd, 624
matDotVec, 625
operator<<, 633, 634
operator*, 626–631
operator*=, 632
operator+, 632
operator+=, 633
Transpose, 635
numvector/nummatrix.h, 612
numvector/numvector.h, 615

offsGam
Vortex2D, 488
offsPos
Vortex2D, 488
oldAirfoil
World2D, 557
oldSheets

Boundary, 91
BoundaryConstLayerAver, 118
Omega
BoundaryConstLayerAver, 115
operator &
nummatrix, 286, 287
nummatrix.h, 613
numvector, 313
operator std::set< P >
numvector, 314
Point2D, 364
operator std::vector< P >
numvector, 315
Point2D, 364
operator std::vector< std::vector< T >>
nummatrix, 288
operator!=
nummatrix, 288
numvector, 315
Point2D, 365
operator<<
defs.h, 585
nummatrix.h, 614
numvector.h, 633, 634
operator*
nummatrix, 289
numvector, 316
numvector.h, 626–631
Point2D, 366
operator*=
nummatrix, 289
numvector, 316
numvector.h, 632
Point2D, 367
operator^
numvector, 324, 325
Point2D, 374, 375
operator()
LogStream, 130
operator+
nummatrix, 290
numvector, 317, 318
numvector.h, 632
Point2D, 367, 368
operator+=
nummatrix, 291
numvector, 318
numvector.h, 633
Point2D, 368
operator-
nummatrix, 291, 292
numvector, 319, 320
Point2D, 370, 371
operator-=
nummatrix, 293
numvector, 320
Point2D, 371
operator/=
nummatrix, 293
numvector, 321
Point2D, 372
operator=
numvector, 321
operator==
nummatrix, 295
numvector, 322
Point2D, 372
operator&
Point2D, 365
operator[]
nummatrix, 296
numvector, 322, 324
Point2D, 373
operator|
nummatrix.h, 615

pStr
LogStream, 131
panelsType
AirfoilParams, 57
parProp, 334
disp, 335
len, 335
myDisp, 336
myLen, 336
totalLen, 336
Parallel, 330
commWork, 333
myidWork, 333
nProcWork, 334
SplitMPIone, 332
SplitMPI, 331
parallel
Queue, 412
World2D, 557
Parallel/Parallel.cpp, 635
Parallel/Parallel.h, 636
ParseStream
StreamParser, 431
Passport, 337
~Passport, 340
airfoilParams, 344
airfoilsDir, 345
dir, 345
GetAllParamsFromParser, 340
info, 345
numericalSchemes, 345
Passport, 338
physicalProperties, 345
PrintAllParams, 343
problemName, 346
problemNumber, 346
timeDiscretizationProperties, 346
wakeDiscretizationProperties, 346
wakesDir, 346
passport
Task, 442

World2D, 557
Passport/Passport.cpp, 638
Passport/Passport.h, 638
Phi
Mechanics, 162
MechanicsRigidGivenLaw, 182
MechanicsRigidImmovable, 201
MechanicsRigidOscillMon, 223
MechanicsRigidOscillPart, 245
MechanicsRigidRotateMon, 273
Phi0
Mechanics, 162
MechanicsRigidGivenLaw, 182
MechanicsRigidImmovable, 201
MechanicsRigidOscillMon, 223
MechanicsRigidOscillPart, 245
MechanicsRigidRotateMon, 273
phiAfl
Airfoil, 52
AirfoilRect, 79
PhiOld
Mechanics, 162
MechanicsRigidGivenLaw, 182
MechanicsRigidImmovable, 202
MechanicsRigidOscillMon, 224
MechanicsRigidOscillPart, 245
MechanicsRigidRotateMon, 273
PhysicalProperties, 347
accelCft, 348
addCurrTime, 349
currTime, 351
getCurrTime, 349
nu, 351
rho, 352
setCurrTime, 350
timeAccel, 352
V0, 350
vInf, 352
vRef, 352
physicalProperties
Passport, 345
PI
defs.h, 592
Point2D/Point2D.cpp, 640
Point2D/Point2D.h, 641
Point2D, 353
~Point2D, 358
CreateMpiType, 358
dist2To, 359
distTo, 360
kcross, 360
length, 361
length2, 361
member, 362
mpiPoint2D, 379
norm1, 362
normalize, 363
norminf, 363
operator std::set< P >, 364
operator std::vector< P >, 364
operator!=, 365
operator*, 366
operator*=, 367
operator^, 374, 375
operator+, 367, 368
operator+=, 368
operator-, 370, 371
operator-=, 371
operator/=, 372
operator==, 372
operator&, 365
operator[], 373
Point2D, 356, 357
r, 379
rotateLeft, 376
rotated, 375
size, 377
toSet, 377
toVector, 377
toZero, 378
unit, 378
pos
Vortex2D, 488
PositionOfAirfoilRcm
Mechanics, 156
MechanicsRigidGivenLaw, 176
MechanicsRigidImmovable, 196
MechanicsRigidOscillMon, 217
MechanicsRigidOscillPart, 238
MechanicsRigidRotateMon, 266
prepared
Queue, 412
Preprocessor, 380
~Preprocessor, 384
afterSlashParser, 384
currentParser, 391
inCharLastEscape, 392
inCharParser, 385
inInlineCommentParser, 386
inInlineLastEscape, 392
inMultilineCommentParser, 387
inMultilineLastStar, 392
inStringLastEscape, 392
inStringParser, 388
initialInput, 392
intermediateOutput, 393
normalParser, 389
Preprocessor, 383
processSymbol, 390
resultStream, 393
resultString, 393
Preprocessor/Preprocessor.cpp, 642
Preprocessor/Preprocessor.h, 643
pressure
MeasureVP, 144
PrintAllParams

Passport, 343
PrintHeaderToTextFile
defs.cpp, 572
defs.h, 586
PrintLogoToTextFile
defs.cpp, 573
defs.h, 587
problemName
Passport, 346
problemNumber
Passport, 346
proc
Task, 442
procState
Queue, 413
procStateVar
Queue, 413
processSymbol
Preprocessor, 390

Queue, 394
~Queue, 397
AddTask, 397
commSolving, 408
commStarting, 408
ConstructProcStateVar, 398
currentKvant, 408
finished, 409
flagFinish, 409
groupAll, 409
groupSolving, 409
groupStarting, 410
info, 410
kvantTime, 410
LoadTasksList, 399
myProcState, 411
myProcStateVar, 411
myidAll, 410
nProcAll, 412
nextKvant, 411
numberOfTask, 412
parallel, 412
prepared, 412
procState, 413
procStateVar, 413
Queue, 396
RunConveyer, 401
sizeCommSolving, 413
solving, 414
task, 414
TaskSplit, 402
TaskUpdate, 406
world2D, 414
Queue/Queue.cpp, 644
Queue/Queue.h, 645

r
Airfoil, 52
AirfoilRect, 79
nummatrix, 299
numvector, 330
Point2D, 379
Vortex2D, 486, 487
Rcm
Mechanics, 162
MechanicsRigidGivenLaw, 182
MechanicsRigidImmovable, 202
MechanicsRigidOscillMon, 224
MechanicsRigidOscillPart, 245
MechanicsRigidRotateMon, 273
rcm
Airfoil, 52
AirfoilRect, 79
Rcm0
Mechanics, 162
MechanicsRigidGivenLaw, 182
MechanicsRigidImmovable, 202
MechanicsRigidOscillMon, 224
MechanicsRigidOscillPart, 246
MechanicsRigidRotateMon, 273
RcmOld
Mechanics, 163
MechanicsRigidGivenLaw, 183
MechanicsRigidImmovable, 202
MechanicsRigidOscillMon, 224
MechanicsRigidOscillPart, 246
MechanicsRigidRotateMon, 274
ReadFromFile
Airfoil, 47
AirfoilRect, 73
VirtualWake, 479
Wake, 502
WakeDataBase, 513
ReadParametersFromDictionary
Mechanics, 157
ReadPointsFromFile
MeasureVP, 142
ReadSpecificParametersFromDictionary
Mechanics, 158
MechanicsRigidGivenLaw, 177
MechanicsRigidImmovable, 197
MechanicsRigidOscillMon, 217
MechanicsRigidOscillPart, 239
MechanicsRigidRotateMon, 267
RemoveFar
Wake, 503
RemoveZero
Wake, 504
ReplaceVarsInString
StreamParser, 432
ReserveMemoryForMatrixAndRhs
World2D, 548
Restruct
Wake, 505
resultStream
Preprocessor, 393
resultString

Preprocessor, 393
rho
PhysicalProperties, 352
rhs
World2D, 558
Rotate
Airfoil, 48
AirfoilRect, 75
rotateLeft
numvector, 326
Point2D, 376
rotated
Point2D, 375
RunConveyer
Queue, 401

SaveKadr
Wake, 506
SaveKadrVtk
Wake, 508
saveTXT
TimeDiscretizationProperties, 445
SaveToStream
defs.cpp, 575, 576
defs.h, 588–590
saveVTK
TimeDiscretizationProperties, 445
saveVP
TimeDiscretizationProperties, 445
Scale
Airfoil, 49
AirfoilRect, 76
scale
AirfoilParams, 57
setAccelCoeff
Gpu, 124
setCollapseCoeff
Gpu, 125
setCurrTime
PhysicalProperties, 350
SetDefault
StreamParser, 434
SetLayersDim
Sheet, 417
Sheet, 415
~Sheet, 416
attachedSourceSheet, 417
attachedVortexSheet, 418
freeVortexSheet, 418
SetLayersDim, 417
Sheet, 416
Sheet/Sheet.cpp, 646
Sheet/Sheet.h, 647
sheetDim
Boundary, 92
BoundaryConstLayerAver, 119
sheets
Boundary, 92
BoundaryConstLayerAver, 119
size
nummatrix, 297
numvector, 326
Point2D, 377
sizeCommSolving
Queue, 413
sol
World2D, 558
SolutionToFreeVortexSheetAndVirtualVortex
Boundary, 90
BoundaryConstLayerAver, 116
SolutionToMechanicalSystem
Mechanics, 158
MechanicsRigidGivenLaw, 177
MechanicsRigidImmovable, 197
MechanicsRigidOscillMon, 218
MechanicsRigidOscillPart, 240
MechanicsRigidRotateMon, 268
SolveLinearSystem
World2D, 549
solving
Queue, 414
source
World2D, 558
Source2D
Vortex2D.h, 666
SplitMPIone
Parallel, 332
SplitMPI
Parallel, 331
SplitString
StreamParser, 435
sqr
defs.h, 590
startEndKvant
Task, 443
state
Task, 443
Step
World2D, 550
StreamParser, 418
~StreamParser, 424
database, 439
defaults, 439
get, 424–429
info, 439
ParseStream, 431
ReplaceVarsInString, 432
SetDefault, 434
SplitString, 435
StreamParser, 421–423
StringToVector, 436
switchers, 440
UpperCase, 437
vars, 440
VectorStringToString, 438
StreamParser/StreamParser.cpp, 648
StreamParser/StreamParser.h, 648

StringToVector
StreamParser, 436
switchers
StreamParser, 440

tMomentAccel
MechanicsRigidRotateMon, 274
tRotateAccel
MechanicsRigidRotateMon, 274
Task, 440
~Task, 442
nProc, 442
passport, 442
proc, 442
startEndKvant, 443
state, 443
Task, 441
wTime, 443
task
Queue, 414
Task.h
TaskState, 651
Task/Task.h, 650
TaskSplit
Queue, 402
TaskState
Task.h, 651
TaskUpdate
Queue, 406
tau
Airfoil, 52
AirfoilRect, 79
timeAccel
PhysicalProperties, 352
timeCalcVortexConvVelo
Times, 453
timeCalcVortexDiffVelo
Times, 453
timeCheckInside
Times, 453
TimeDiscretizationProperties, 443
dt, 444
nameLength, 444
saveTXT, 445
saveVTK, 445
saveVP, 445
timeStart, 445
timeStop, 445
timeDiscretizationProperties
Passport, 346
timeFillMatrixAndRhs
Times, 453
timeGetHydroDynamForce
Times, 453
timeMoveVortexes
Times, 453
timeOther
Times, 454
timePeriod
defs.h, 581
timeReserveMemoryForMatrixAndRhs
Times, 454
timeRestruct
Times, 454
timeSaveKadr
Times, 454
timeSolveLinearSystem
Times, 454
timeStart
TimeDiscretizationProperties, 445
timeStop
TimeDiscretizationProperties, 445
timeVP
Times, 455
timeWakeSort
Times, 455
timeWholeStep
Times, 455
Times, 446
~Times, 448
dT, 449
GenerateStatHeader, 449
GenerateStatString, 450
timeCalcVortexConvVelo, 453
timeCalcVortexDiffVelo, 453
timeCheckInside, 453
timeFillMatrixAndRhs, 453
timeGetHydroDynamForce, 453
timeMoveVortexes, 453
timeOther, 454
timeReserveMemoryForMatrixAndRhs, 454
timeRestruct, 454
timeSaveKadr, 454
timeSolveLinearSystem, 454
timeVP, 455
timeWakeSort, 455
timeWholeStep, 455
Times, 448
ToZero, 451, 452
W, 455
Times/Times.cpp, 652
Times/Times.h, 653
timestat
World2D, 558
toIdentity
nummatrix, 297
toSet
numvector, 327
Point2D, 377
toVector
numvector, 327
Point2D, 377
ToZero
Times, 451, 452
toZero
nummatrix, 298
numvector, 327

Point2D, 378
totalLen
parProp, 336
Transpose
numvector.h, 635
transpose
nummatrix, 298

u
MechanicsRigidOscillPart, 246
u0
MechanicsRigidOscillPart, 246
uOld
MechanicsRigidOscillPart, 246
unit
numvector, 328
Point2D, 378
upRight
Airfoil, 53
AirfoilRect, 79
UpperCase
StreamParser, 437

v
Airfoil, 53
AirfoilRect, 80
V0
PhysicalProperties, 350
vInf
PhysicalProperties, 352
VM2D.cpp
CreateMpiTypes, 662
Initializers, 662
main, 663
VM2D/VM2D.cpp, 661
vRef
PhysicalProperties, 352
vars
StreamParser, 440
Vcm
Mechanics, 163
MechanicsRigidGivenLaw, 183
MechanicsRigidImmovable, 202
MechanicsRigidOscillMon, 224
MechanicsRigidOscillPart, 247
MechanicsRigidRotateMon, 274
Vcm0
Mechanics, 163
MechanicsRigidGivenLaw, 183
MechanicsRigidImmovable, 203
MechanicsRigidOscillMon, 225
MechanicsRigidOscillPart, 247
MechanicsRigidRotateMon, 274
VcmOld
Mechanics, 163
MechanicsRigidGivenLaw, 183
MechanicsRigidImmovable, 203
MechanicsRigidOscillMon, 225
MechanicsRigidOscillPart, 247
MechanicsRigidRotateMon, 275
vecHalfGamma
VirtualWake, 481
VectorStringToString
StreamParser, 438
VeloOfAirfoilPanels
Mechanics, 158
MechanicsRigidGivenLaw, 178
MechanicsRigidImmovable, 198
MechanicsRigidOscillMon, 219
MechanicsRigidOscillPart, 241
MechanicsRigidRotateMon, 269
VeloOfAirfoilRcm
Mechanics, 159
MechanicsRigidGivenLaw, 179
MechanicsRigidImmovable, 198
MechanicsRigidOscillMon, 220
MechanicsRigidOscillPart, 242
MechanicsRigidRotateMon, 269
Velocity, 456
~Velocity, 458
CalcConvVelo, 458
CalcDiffVelo, 459
CalcDiffVeloI1I2ToSetOfPointsFromPanels, 461
Velocity, 457
virtualVortexesParams, 462
W, 462
wakeVortexesParams, 462
velocity
MeasureVP, 144
World2D, 558
Velocity/Velocity.cpp, 654
Velocity/Velocity.h, 654
Velocity/VelocityBiotSavart.cpp, 656
Velocity/VelocityBiotSavart.h, 657
VelocityBiotSavart, 463
~VelocityBiotSavart, 465
CalcConvVelo, 465
CalcConvVeloToSetOfPoints, 466
CalcDiffVelo, 469
CalcDiffVeloI1I2ToSetOfPoints, 471
CalcDiffVeloI1I2ToSetOfPointsFromPanels, 473
VelocityBiotSavart, 464
virtualVortexesParams, 476
W, 476
wakeVortexesParams, 476
VelocityBiotSavart.cpp
ModifyE2, 656
velocityComputation
NumericalSchemes, 277
virtVortParams
Mechanics, 163
MechanicsRigidGivenLaw, 183
MechanicsRigidImmovable, 203
MechanicsRigidOscillMon, 225
MechanicsRigidOscillPart, 247

MechanicsRigidRotateMon, 275
virtualVortexesParams
Velocity, 462
VelocityBiotSavart, 476
VirtualWake, 477
~VirtualWake, 479
aflPan, 480
bnd, 480
ReadFromFile, 479
vecHalfGamma, 481
VirtualWake, 478
vtx, 481
W, 481
virtualWake
Boundary, 92
BoundaryConstLayerAver, 119
VirtualWake/VirtualWake.cpp, 659
VirtualWake/VirtualWake.h, 659
VirtualWakeSynchronize
Параллельные функции, 24
viscousStress
Airfoil, 53
AirfoilRect, 80
Vortex2D.h
Source2D, 666
Vortex2D/Vortex2D.cpp, 664
Vortex2D/Vortex2D.h, 665
Vortex2D, 482
~Vortex2D, 484
CreateMpiType, 484
g, 485, 486
gam, 487
mpiVortex2D, 487
offsGam, 488
offsPos, 488
pos, 488
r, 486, 487
Vortex2D, 483
vortexPerPanel
Boundary, 92
BoundaryConstLayerAver, 119
WakeDiscretizationProperties, 518
VortexesParams, 489
convVelo, 490
diffVelo, 490
epsastWake, 490
I0, 490
I1, 491
I2, 491
I3, 491
vtx
VirtualWake, 481
Wake, 511
WakeDataBase, 514

W
Airfoil, 53
AirfoilRect, 80
Boundary, 93
BoundaryConstLayerAver, 120
Gpu, 125
MeasureVP, 144
Mechanics, 164
MechanicsRigidGivenLaw, 184
MechanicsRigidImmovable, 203
MechanicsRigidOscillMon, 225
MechanicsRigidOscillPart, 247
MechanicsRigidRotateMon, 275
Times, 455
Velocity, 462
VelocityBiotSavart, 476
VirtualWake, 481
Wake, 511
WakeDataBase, 515
wTime
Task, 443
Wake, 492
~Wake, 494
Collaps, 495
collapseRightBorderParameter, 510
collapseScaleParameter, 510
GetPairs, 497
MoveInside, 499
MoveInsideMovingBoundary, 501
neighb, 510
ReadFromFile, 502
RemoveFar, 503
RemoveZero, 504
Restruct, 505
SaveKadr, 506
SaveKadrVtk, 508
vtx, 511
W, 511
Wake, 494
wake
World2D, 559
Wake/Wake.cpp, 667
Wake/Wake.h, 667
WakeDataBase, 511
~WakeDataBase, 513
ReadFromFile, 513
vtx, 514
W, 515
WakeDataBase, 512
WakeDataBase/WakeDataBase.cpp, 669
WakeDataBase/WakeDataBase.h, 669
WakeDiscretizationProperties, 515
delta, 516
distFar, 516
eps, 517
eps2, 517
epscol, 517
fileSource, 517
fileWake, 517
vortexPerPanel, 518
wakeDiscretizationProperties
Passport, 346

wakeMotionIntegrator
NumericalSchemes, 277
WakeSynchronize
Параллельные функции, 25
wakeVortexesParams
Velocity, 462
VelocityBiotSavart, 476
wakesDir
Passport, 346
Wcm
Mechanics, 164
MechanicsRigidGivenLaw, 184
MechanicsRigidImmovable, 203
MechanicsRigidOscillMon, 225
MechanicsRigidOscillPart, 248
MechanicsRigidRotateMon, 275
Wcm0
Mechanics, 164
MechanicsRigidGivenLaw, 184
MechanicsRigidImmovable, 204
MechanicsRigidOscillMon, 226
MechanicsRigidOscillPart, 248
MechanicsRigidRotateMon, 275
WcmOld
Mechanics, 164
MechanicsRigidGivenLaw, 184
MechanicsRigidImmovable, 204
MechanicsRigidOscillMon, 226
MechanicsRigidOscillPart, 248
MechanicsRigidRotateMon, 276
World2D/World2D.cpp, 671
World2D/World2D.h, 671
World2D, 518
~World2D, 523
airfoil, 555
boundary, 555
CalcVortexVelo, 524
CheckInside, 530
cuda, 556
currentStep, 556
FillMatrixAndRhs, 531
GenerateMechanicsHeader, 534
getAirfoil, 534
getBoundary, 535
getCuda, 536
getCurrentStep, 536
getInfo, 537, 538
getNonConstAirfoil, 539
getNonConstBoundary, 539
getNonConstVelocity, 540
getNumberOfAirfoil, 540
getNumberOfBoundary, 541
getParallel, 541
getPassport, 542
getSource, 543
getTimestat, 544
getVelocity, 545
getWake, 545
info, 556
invMatr, 556
isFinished, 546
matr, 556
measureVP, 557
mechanics, 557
MoveVortexes, 547
oldAirfoil, 557
parallel, 557
passport, 557
ReserveMemoryForMatrixAndRhs, 548
rhs, 558
sol, 558
SolveLinearSystem, 549
source, 558
Step, 550
timestat, 558
velocity, 558
wake, 559
World2D, 521
world2D
Queue, 414

y
MechanicsRigidOscillPart, 248
y0
MechanicsRigidOscillPart, 248
yOld
MechanicsRigidOscillPart, 248