

代码结构


Chrono 源码 找到主函数以及运行流程

- 1. 主体结构和模块导入
- 2. 全局变量（仿真模式控制）
- 3. 主函数入口
- 4. 仿真参数设置
- 5. 创建系统对象
- 6. 创建材料（用于物体接触）
- 7. 创建球体对象（动态刚体）
- 8. 创建地面容器（固定刚体）
- 9. 初始化可视化引擎
- 10. 仿真主循环

Demo使用APGD求解器

Chrono Source Code

代码结构

文件/文件夹名	说明
cmake/	包含 CMake 模块或宏定义，例如用于检测第三方库、路径等配置脚本。
contrib/	外部贡献的脚本、模块或集成工具，例如 VSG 图形支持。
data/	仿真所需的数据资源，例如地形、车辆参数、3D 模型等。
docs/	项目的说明文档或用户手册。
doxygen/	Doxygen 文档生成配置，支持自动生成 C++ API 文档。
src/	 主要源代码所在位置。真正的 Chrono 模块（如刚体、碰撞、求解器等）都在这里。

template_project*	不同语言/应用场景的模板项目（C++/ROS/C#/FMU 等），供用户快速开始。
.clang-format	C++ 代码自动格式化配置，统一代码风格。
.gitignore	Git 忽略不需要上传的文件（如 build/、中间文件等）。
CMakeLists.txt	★ 项目主 CMake 构建脚本入口，描述如何构建整个 Chrono。
CMakePresets.json	定义了一些推荐的构建预设（比如启用GPU、编译示例、调试模式等）。
CTestConfig.cmake	配置测试套件（如单元测试）。
LICENSE	开源协议（BSD-3-Clause）。
CHANGELOG.md	更新日志，记录项目每个版本变更点。

Chrono 源码 找到主函数以及运行流程

随意找一个demo文件来看他的运行流程以及主函数

1. 主体结构 and 模块导入

```
1 #include "chrono/physics/ChSystem.h"
2 #include "chrono/core/ChRealtimeStep.h"
3 #include "chrono/assets/ChVisualSystem.h"
```

导入了 Chrono 核心模块（系统、实时步进器、可视化模块）；

根据条件编译导入：

```
1 #include "chrono_irrlicht/ChVisualSystemIrrlicht.h"
2 #include "chrono_vsg/ChVisualSystemVSG.h"
```

表示可选的可视化系统：`Irrlicht` 或 `VSG`。

2. 全局变量（仿真模式控制）

```
1 ChContactMethod contact_method = ChContactMethod::SMC; // 选择接触模型
2 ChVisualSystem::Type vis_type = ChVisualSystem::Type::VSG; // 可视化引擎
3 ChCollisionSystem::Type coll_type = ChCollisionSystem::Type::BULLET; // 碰撞系统
```

3. 主函数入口

主函数从这里开始。打印版权和版本信息。

4. 仿真参数设置

```
1 double gravity = -9.81;
2 double time_step = contact_method == ChContactMethod::NSC ? 1e-3 : 1e-5;
3 double render_fps = 100;
4 ...
```

- 设置重力；
- 根据接触模型设置时间步长（SMC 需要更小的步长）；
- 设置渲染帧率；
- 掉落球体参数（Parameters for the falling ball）；
- 承接地板参数（Parameters for the containing bin）

5. 创建系统对象

1. 创建 Chrono 系统对象
2. 设置重力
3. 设置碰撞系统类型

6. 创建材料（用于物体接触）

```
1 ChContactMaterialData mat_data;
2 mat_data.mu = 0.4f;    // 摩擦系数
3 mat_data.cr = 0.1f;    // 恢复系数
4 auto material = mat_data.CreateMaterial(contact_method);
```

7. 创建球体对象（动态刚体）

```
1 auto ball = chrono_types::make_shared<ChBody>();
2 ball->SetMass(mass);
3 ball->SetInertiaXX(...); // 设置转动惯量
4 ball->SetPos(pos);
5 ball->SetRot(rot);
```

```
6 ball->SetPosDt(init_vel);
7 ball->SetFixed(false);
```

- 设置质量、惯量、位置、速度；
- 设置为可移动。

添加碰撞形状和视觉形状

8. 创建地面容器（固定刚体）

9. 初始化可视化引擎

10. 仿真主循环

我们主要围绕求解器chrono中的源码部分，src/

[chrono/src/chrono/solver/ChSolverAPGD.cpp](#) and [ChSolverAPGD.h](#)

APGD（Accelerated Projected Gradient Descent，加速投影梯度下降法）专门用于求解非平滑动力学系统中的约束问题。（也就是有friction）

算法讲解

在 Chrono 的多体动力学中，考虑有接触和摩擦时，系统约束力满足如下 **Cone Complementarity Problem**：

$$\text{Find } \lambda \in \mathcal{K} \text{ such that: } N\lambda + r \in \mathcal{K}^*, \text{ and } \langle \lambda, N\lambda + r \rangle = 0$$

符号	含义	备注
$\lambda \in \mathbb{R}^n$	接触/摩擦力（拉格朗日乘子）	就是最终要求解的量
$\mathcal{K} \subset \mathbb{R}^n$	锥形可行集（如摩擦锥）	限制 λ 必须满足的方向与大小条件
\mathcal{K}^*	对偶锥（dual cone）	用于描述残差的合法性

$N = D^T M^{-1} D$	Schur 补矩阵（系统刚度）	正定或半正定
$r = D^T M^{-1} k + c$	右边项（力平衡与约束项）	通常表示“违约量”或者“应力源”
D	约束雅可比矩阵	每个约束对状态变量的线性化
M	质量矩阵（对角或块对角）	多体系统质量项
k	外力项（包括重力、驱动等）	输入激励部分
c	约束漂移项（如 ϕ/h ）	时间离散带来的误差补偿项

如何把这个 CCP 转成 QP 问题

Step 1: 互补条件等价于最优性条件

互补条件：

$$\lambda \in \mathcal{K}, \quad w = N\lambda + r \in \mathcal{K}^*, \quad \langle \lambda, w \rangle = 0$$

这正好等价于一个 QP 的一阶 KKT 条件（Karush-Kuhn-Tucker optimality conditions）。

Step 2: 构造 QP 优化问题

构造如下目标函数：

$$\min_{\lambda \in \mathcal{K}} \quad f(\lambda) = \frac{1}{2} \lambda^T N \lambda + r^T \lambda$$

✅ 为什么这个目标函数成立？

我们求解的这个函数 $f(\lambda)$ 是严格凸的二次函数（因为 N 是正定/半正定），且有以下性质：

- 梯度：

$$\nabla f(\lambda) = N\lambda + r =: w$$

- 最优性条件（投影子梯度法）为：

$$\lambda \in \mathcal{K}, \quad w = \nabla f(\lambda) \in \mathcal{K}^*, \quad \langle \lambda, w \rangle = 0$$

正是我们 CCP 的定义！因此，这个 QP 问题与 CCP 是等价的。

Step 3: 将其代入 Chrono 中的变量
具体形式如下:

$$\min_{\lambda \in \mathcal{K}} \underbrace{\frac{1}{2} \lambda^T (D^T M^{-1} D) \lambda}_{\text{刚度能}} + \underbrace{(D^T M^{-1} k + c)^T \lambda}_{\text{力功项}}$$

其中:

- D : 约束对状态变量的影响
- M : 质量矩阵
- k : 非约束力
- c : 约束偏移项, 通常是 ϕ/h

Project Chrono 中 `ChSolverAPGD` 的实现代码

```
1 // =====
2 // PROJECT CHRONO - http://projectchrono.org
3 //
4 // Copyright (c) 2014 projectchrono.org
5 // All rights reserved.
6 //
7 // Use of this source code is governed by a BSD-style license that can be found
8 // in the LICENSE file at the top level of the distribution and at
9 // http://projectchrono.org/license-chrono.txt.
10 //
11 // =====
12 // Authors: Alessandro Tasora, Radu Serban
13 // =====
14
15 #include "chrono/solver/ChSolverAPGD.h"
16
17 #include <iostream>
18 #include <sstream>
19 #include <string>
20 #include <valarray>
21 #include <vector>
22
23 namespace chrono {
24
25 // Register into the object factory, to enable run-time dynamic creation and
26 persistence
27 //CH_FACTORY_REGISTER(...): 用于 Chrono 的工厂机制, 支持 run-time 动态创建该类对
```

```

28 象。
29 CH_FACTORY_REGISTER(ChSolverAPGD)
30
31 //构造函数初始化两个变量: nc: 约束个数 residual: 当前残差 (用于迭代终止判断)
32 ChSolverAPGD::ChSolverAPGD() : nc(0), residual(0.0) {}
33
34 //SchurBvectorCompute: 构造右端项 b 详见注释 (1)
35 void ChSolverAPGD::SchurBvectorCompute(ChSystemDescriptor& sysd) {
36     // ***TO DO*** move the following thirty lines in a short function
37     ChSystemDescriptor::SchurBvectorCompute() ?
38
39     // Compute the b_schur vector in the Schur complement equation  $N*1 = b\_schur$ 
40     // with
41     //  $N\_schur = D' * (M^{-1}) * D$ 
42     //  $b\_schur = -c + D' * (M^{-1}) * k = b\_i + D' * (M^{-1}) * k$ 
43     // but flipping the sign of lambdas,  $b\_schur = -b\_i - D' * (M^{-1}) * k$ 
44     // Do this in three steps:
45
46     // Put  $(M^{-1}) * k$  in q sparse vector of each variable..
47     for (unsigned int iv = 0; iv < sysd.GetVariables().size(); iv++)
48         if (sysd.GetVariables()[iv]->IsActive())
49             sysd.GetVariables()[iv]-
50 >ComputeMassInverseTimesVector(sysd.GetVariables()[iv]->State(),
51
52 sysd.GetVariables()[iv]->Force()); //  $q = [M]^{-1} * f_b$ 
53
54     // ...and now do  $b\_schur = -D' * q = -D' * (M^{-1}) * k$  ..
55     r.setZero();
56     int s_i = 0;
57     for (unsigned int ic = 0; ic < sysd.GetConstraints().size(); ic++)
58         if (sysd.GetConstraints()[ic]->IsActive()) {
59             r(s_i, 0) = sysd.GetConstraints()[ic]->ComputeJacobianTimesState();
60             ++s_i;
61         }
62
63     // ..and finally do  $b\_schur = b\_schur - c$ 
64     sysd.BuildBiVector(tmp); //  $b\_i = -c = \phi/h$ 
65     r += tmp;
66 }
67
68 //Res4(): 计算当前残差 (优化目标) 详见注释 (2)
69 double ChSolverAPGD::Res4(ChSystemDescriptor& sysd) {
70     // Project the gradient (for rollback strategy)
71     //  $g\_proj = (1 - \text{project\_orthogonal}(1 - gdiff * g, fric)) / gdiff$ ;
72     double gdiff = 1.0 / (nc * nc);
73     sysd.SchurComplementProduct(tmp, gammaNew); //  $tmp = N * gammaNew$ 
74     tmp = gammaNew - gdiff * (tmp + r); // Note: no aliasing issues
75     here
76     sysd.ConstraintsProject(tmp); // tmp =
77     ProjectionOperator(gammaNew - gdiff * g)
78     tmp = (gammaNew - tmp) / gdiff; // Note: no aliasing issues
79     here
80

```

```

81     return tmp.norm();
82 }
83
84 //Solve() 主体: APGD 算法流程
85 double ChSolverAPGD::Solve(ChSystemDescriptor& sysd) {
86     const std::vector<ChConstraint*>& mconstraints = sysd.GetConstraints();
87     const std::vector<ChVariables*>& mvariables = sysd.GetVariables();
88     if (verbose)
89         std::cout << "Number of constraints: " << mconstraints.size()
90         << "\nNumber of variables : " << mvariables.size() <<
91     std::endl;
92
93     // Update auxiliary data in all constraints before starting,
94     // that is:  $g_i = [Cq_i] * [invM_i] * [Cq_i]'$  and  $[Eq_i] = [invM_i] * [Cq_i]'$ 
95     for (unsigned int ic = 0; ic < mconstraints.size(); ic++)
96         mconstraints[ic]->Update_auxiliary();
97
98     double L, t;
99     double theta;
100    double thetaNew;
101    double Beta;
102    double obj1, obj2;
103
104    nc = sysd.CountActiveConstraints();
105    gamma_hat.resize(nc);
106    gammaNew.resize(nc);
107    g.resize(nc);
108    y.resize(nc);
109    gamma.resize(nc);
110    yNew.resize(nc);
111    r.resize(nc);
112    tmp.resize(nc);
113
114    residual = 10e30;
115
116    Beta = 0.0;
117    obj1 = 0.0;
118    obj2 = 0.0;
119
120    // Compute the b_schur vector in the Schur complement equation  $N*1 = b\_schur$ 
121    SchurBvectorCompute(sysd);
122
123    // If no constraints, return now. Variables contain  $M^{-1} * f$  after call to
124    SchurBvectorCompute.
125    // This early exit is needed, else we get division by zero and a potential
126    infinite loop.
127    if (nc == 0) {
128        return 0;
129    }
130
131    // Optimization: backup the q sparse data computed above,
132    // because  $(M^{-1})^k$  will be needed at the end when computing primals.
133    ChVectorDynamic<> Minvk;

```



```

134     sysd.FromVariablesToVector(Minvk, true);
135
136     // (1) gamma_0 = zeros(nc,1)
137     if (m_warm_start) {
138         for (unsigned int ic = 0; ic < mconstraints.size(); ic++)
139             if (mconstraints[ic]->IsActive())
140                 mconstraints[ic]->IncrementState(mconstraints[ic]-
141 >GetLagrangeMultiplier());
142     } else {
143         for (unsigned int ic = 0; ic < mconstraints.size(); ic++)
144             mconstraints[ic]->SetLagrangeMultiplier(0.);
145     }
146     sysd.FromConstraintsToVector(gamma);
147
148     // (2) gamma_hat_0 = ones(nc,1)
149     gamma_hat.setConstant(1.0);
150
151     // (3) y_0 = gamma_0
152     y = gamma;
153
154     // (4) theta_0 = 1
155     theta = 1.0;
156
157     // (5) L_k = norm(N * (gamma_0 - gamma_hat_0)) / norm(gamma_0 - gamma_hat_0)
158     tmp = gamma - gamma_hat;
159     L = tmp.norm();
160     sysd.SchurComplementProduct(yNew, tmp, nullptr); // yNew = N * tmp = N *
161 (gamma - gamma_hat)
162     L = yNew.norm() / L;
163     yNew.setZero(); // RADU is this really necessary here?
164
165     // (6) t_k = 1 / L_k
166     t = 1.0 / L;
167
168     // RADU
169     // Check correctness (e.g. sign of 'r' in comments vs. code)
170
171     std::fill(violation_history.begin(), violation_history.end(), 0.0);
172     std::fill(dlambd_history.begin(), dlambd_history.end(), 0.0);
173
174     // (7) for k := 0 to N_max
175     for (m_iterations = 0; m_iterations < m_max_iterations; m_iterations++) {
176         // (8) g = N * y_k - r
177         // (9) gamma_(k+1) = ProjectionOperator(y_k - t_k * g)
178         sysd.SchurComplementProduct(g, y); // g = N * y
179         gammaNew = y - t * (g + r);
180         sysd.ConstraintsProject(gammaNew);
181
182         // (10) while 0.5 * gamma_(k+1)' * N * gamma_(k+1) - gamma_(k+1)' * r >=
183         //         0.5 * y_k' * N * y_k - y_k' * r + g' * (gamma_(k+1) - y_k)
184 + 0.5 * L_k * norm(gamma_(k+1) - y_k)^2
185         sysd.SchurComplementProduct(tmp, gammaNew); // tmp = N * gammaNew;
186         obj1 = gammaNew.dot(0.5 * tmp + r);

```

```

187
188     sysd.SchurComplementProduct(tmp, y); // tmp = N * y;
189     obj2 = y.dot(0.5 * tmp + r) + (gammaNew - y).dot(g + 0.5 * L * (gammaNew
190 - y));
191
192     while (obj1 >= obj2) {
193         // (11) L_k = 2 * L_k
194         L = 2.0 * L;
195
196         // (12) t_k = 1 / L_k
197         t = 1.0 / L;
198
199         // (13) gamma_(k+1) = ProjectionOperator(y_k - t_k * g)
200         gammaNew = y - t * g;
201         sysd.ConstraintsProject(gammaNew);
202
203         // Update obj1 and obj2
204         sysd.SchurComplementProduct(tmp, gammaNew); // tmp = N * gammaNew;
205         obj1 = gammaNew.dot(0.5 * tmp + r);
206
207         sysd.SchurComplementProduct(tmp, y); // tmp = N * y;
208         obj2 = y.dot(0.5 * tmp + r) + (gammaNew - y).dot(g + 0.5 * L *
209 (gammaNew - y));
210     } // (14) endwhile
211
212     // (15) theta_(k+1) = (-theta_k^2 + theta_k * sqrt(theta_k^2 + 4)) / 2
213     thetaNew = (-theta * theta + theta * std::sqrt(theta * theta + 4.0)) /
214 2.0;
215
216     // (16) Beta_(k+1) = theta_k * (1 - theta_k) / (theta_k^2 + theta_(k+1))
217     Beta = theta * (1.0 - theta) / (theta * theta + thetaNew);
218
219     // (17) y_(k+1) = gamma_(k+1) + Beta_(k+1) * (gamma_(k+1) - gamma_k)
220     yNew = gammaNew + Beta * (gammaNew - gamma);
221
222     // (18) r = r(gamma_(k+1))
223     double res = Res4(sysd);
224
225     if (res < residual) { // (19) if r < epsilon_min
226         residual = res; // (20) r_min = r
227         gamma_hat = gammaNew; // (21) gamma_hat = gamma_(k+1)
228     } // (22) endif
229
230     if (residual < m_tolerance) { // (23) if r < Tau
231         break; // (24) break
232     } // (25) endif
233
234     if (g.dot(gammaNew - gamma) > 0) { // (26) if g' * (gamma_(k+1) -
235 gamma_k) > 0
236         yNew = gammaNew; // (27) y_(k+1) = gamma_(k+1)
237         thetaNew = 1.0; // (28) theta_(k+1) = 1
238     } // (29) endif
239

```

```

240         // (30)  $L_k = 0.9 * L_k$ 
241          $L = 0.9 * L;$ 
242
243         // (31)  $t_k = 1 / L_k$ 
244          $t = 1.0 / L;$ 
245
246         // perform some tasks at the end of the iteration
247         if (this->record_violation_history) {
248             AtIterationEnd(residual, (gammaNew - gamma).lpNorm<Eigen::Infinity>
249             ()), m_iterations);
250         }
251
252         // Update iterates
253         theta = thetaNew;
254         gamma = gammaNew;
255         y = yNew;
256     } // (32) endfor
257
258     if (verbose)
259         std::cout << "Residual: " << residual << ", Iter: " << m_iterations <<
260     std::endl;
261
262     // (33) return Value at time step  $t_{l+1}$ ,  $\gamma_{l+1} := \gamma_{\text{hat}}$ 
263     sysd.FromVectorToConstraints(gamma_hat);
264
265     // Resulting PRIMAL variables:
266     // compute the primal variables as  $v = (M^{-1})(k + D*1)$ 
267     //  $v = (M^{-1}) * k \dots$  (by rewinding to the backup vector computed at the
268     beginning)
269     sysd.FromVectorToVariables(Minvk);
270
271     //  $\dots + (M^{-1}) * D * 1$  (this increment and also stores 'qb' in the
272     ChVariable items)
273     for (size_t ic = 0; ic < mconstraints.size(); ic++) {
274         if (mconstraints[ic]->IsActive())
275             mconstraints[ic]->IncrementState(mconstraints[ic]-
276             >GetLagrangeMultiplier());
277     }
278
279     return residual;
280 }
281
282 void ChSolverAPGD::Dump_Rhs(std::vector<double>& temp) {
283     for (int i = 0; i < r.size(); i++) {
284         temp.push_back(r(i));
285     }
286 }
287
288 void ChSolverAPGD::Dump_Lambda(std::vector<double>& temp) {
289     for (int i = 0; i < gamma_hat.size(); i++) {
290         temp.push_back(gamma_hat(i));
291     }
292 }

```

```
}  
  
} // end namespace chrono
```

注释 (1):

求解下列 CCP 问题所需的右边项:

$$\mathbf{b}_{\text{schur}} = -D^T M^{-1} \mathbf{k} - \mathbf{c}$$

三步逻辑:

1. 计算 $M^{-1} \mathbf{k}$ 并存入每个变量内部状态中。
2. 乘以 D^T , 得到 $D^T M^{-1} \mathbf{k}$
3. 加上 $-\mathbf{c}$ (由 `BuildBiVector` 构造)

结果保存在成员变量 `r` 中 (表示残差向量)。

注释 (2):

数学意义:

残差的定义如下 (类似于一阶最优性误差):

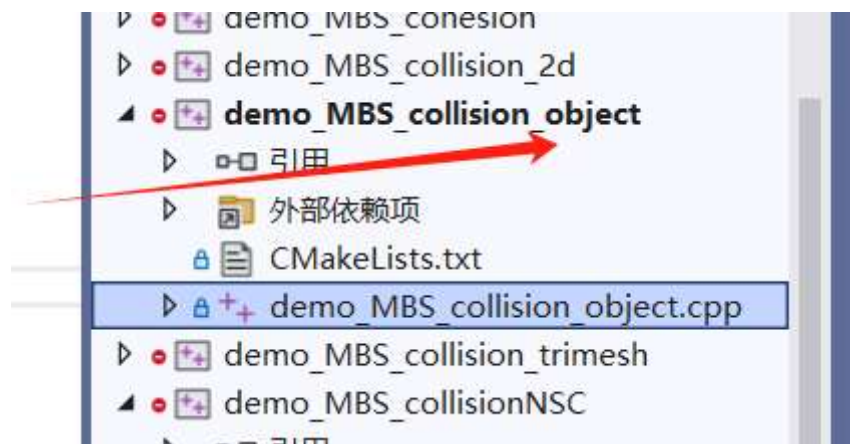
$$\text{residual} = \left\| \frac{1}{g_{\text{diff}}} (\gamma - \text{Proj}(\gamma - g_{\text{diff}} \cdot g)) \right\|$$

用于判断当前 γ 是否足够逼近最优点。

`chrono/src/chrono/physics/ChSystem.h`

它定义了整个 **Chrono 仿真系统的中枢控制类**, 是所有动力学模拟的起点。

Demo使用APGD求解器



demo_MBS_collision_object

- NSC (Non-Smooth Contact) 方法;
- APGD 求解器;
- Multicore 或 Bullet 碰撞系统;
- Irrlicht 可视化;
- 自定义 `ContactManager` 打印接触信息。

