# MinGLE：极简、可配置且具有教学性的 Geant4 应用程序模板

刘静

南达科他大学物理系，美国南达科他州弗米利恩市东克拉克街*414*号，邮编*57069*

arXiv:2601.08469v1 [hep-ex] 13 Jan 2026

**摘要**

Geant4 工具包是模拟粒子在物质中输运的行业标准，但其陡峭的学习曲线常常令新手开发者望而却步。本文介绍 MinGLE（Geant4迷你学习示例），这是一款极简应用程序模板，旨在帮助用户从 Geant4终端用户成长为开发者。MinGLE 利用现代工厂类、文本几何语法和通用粒子源，在不到70行 C++ code 代码中提供了一个完整、可运行的模拟内核。其教学创新之处在于采用循序渐进的开发路线图，并通过十个可测试的 Git 分支和标签加以规范。每个阶段都会引入一个核心 Geant4 组件，涵盖用户界面、物理、计分和磁场，确保每一行代码都对应着明确的需求。MinGLE 既是新项目强大而通用的起点，也是循序渐进的教育资源，能够将被动阅读文档转变为积极的动手开发体验。

关键词：Geant4、M onte Carlo 模拟、应用程序模板、文本几何、教学法、Git、GitHub、极简设计 gn

## 1. 引言

Geant4 [1–3] 是用于模拟粒子穿过物质过程的标准软件，在各个科学学科的探测器设计和数据分析中发挥着关键作用。Geant4 明确将自身定位为工具包，要求用户充当 Geant4 应用程序开发者而非最终用户，因为用户必须理解并正确组装多个核心 Geant4 组件（运行管理、物理、几何等），才能创建可正常运行的可执行文件。因此，Geant4 的初始学习曲线十分陡峭。

官方 Geant4示例的主要设计目的是展示各种 Geant4 组件的组装方式，但通常与特定的探测器模型紧密相关，因此若要将其拆解为通用模板，往往需要付出大量努力。

为了降低这一入门门槛，诸如功能丰富但占用空间小的 Geant4 示例应用程序（GEARS） [4]之类的项目试图使用户完全不必进行 C++ 开发。通过允许用户借助外部文本文件对模拟进行全面配置，GEARS 使初学者能够作为预构建应用程序的最终用户进行操作，而不是作为开发者从头开始组装工具包。

Geant4合作组也认识到，传统上依赖复杂的低层组件组装容易出错，并且要求用户掌握过多的具体知识。现代组件（例如用于物理列表和运行管理器的工厂类）提供了更简洁的抽象接口，有效地在内部完成了部分组装工作。然而，这些类仍然是工具包的组件。它们仍需在 C++ code 中显式调用和组装，而这并不能像 GEARS 一样，将 Geant4 转变为对最终用户友好的应用程序。

本文介绍 MinGLE（Mini Geant4 Learning Example, Geant4 迷你学习示例） [5], ，这是一种应用程序模板，旨在弥合 Geant4终端用户与 Geant4应用程序开发者之间的差距。MinGLE 与 GEARS 类似，所有必要的配置都通过宏文件和文本文件在运行时完成。然而，与 GEARS 不同，MinGLE 不包含任何 Geant4 扩展代码；它完全依赖标准 Geant4 组件和接口。

MinGLE 有两个主要用途：

- 为偏好简约、标准 Geant4 应用程序核心的开发者提供一个简洁、易用的起点。
- 为熟悉运行时配置（例如，通过使用 GEARS）的终端用户提供过渡性学习工具，帮助他们进一步了解并修改底层 C++源代码。

MinGLE 提供了一个完整、现代的 Geant4 应用程序模板，*mingle.cc*，其 C 代码不足 70 行++，完整展示于清单1中。它通过利用现代 Geant4 工厂类和外部配置文件实现了这种简约设计，确保程序具有高度可配置性、通用性，且易于学习。

值得注意的是，尽管 MinGLE 主要作为可扩展的模板提供，但它本身已包含执行各种模拟和分析任务所需的全部 Geant4 核心组件。

## 2. 代码管理

MinGLE 托管在 GitHub 上 [6] ，以利用现代软件开发生态系统，并使用 Git [7] 和 GitHub

电子邮箱地址：jing.liu@usd.edu（刘静）

```
1 #include <G4VUserDetectorConstruction.hh>2 #include <G4GlobalMagFieldMessenger.hh>3 #include <
G4tgbVolumeMgr.hh>4 #include <G4AutoDelete.hh>5 class Detector : public G4VUserDetectorConstruction 6 { 7
public: 8 G4VPhysicalVolume* Construct() { 9 G4tgbVolumeMgr::GetInstance()->AddTextFile("detector.tg"); 10
return G4tgbVolumeMgr::GetInstance()->ReadAndConstructDetector(); 11 } ///<从文本文件 "detector.tg" 加载探测器定
义 12 void ConstructSDandField() { 13 G4AutoDelete::Register(new G4GlobalMagFieldMessenger()); 14 } ///<启用 /
globalField/ 以设置均匀B场 15 }; 16 17 #include <G4VUserPrimaryGeneratorAction.hh>18 #include <
G4GeneralParticleSource.hh>19 class Generator : public G4VUserPrimaryGeneratorAction 20 { 21 private: 22
G4GeneralParticleSource* fGPS; 23 public: 24 Generator() : G4VUserPrimaryGeneratorAction() { fGPS =new
G4GeneralParticleSource; }25 ~Generator() { delete fGPS; } 26 void GeneratePrimaries(G4Event *evt) { fGPS->
GeneratePrimaryVertex(evt); } 27 }; 28 29 #include <G4VUserActionInitialization.hh>30 class Action : public
G4VUserActionInitialization 31 { 32 public: 33 void Build() const { SetUserAction(new Generator); } 34 }; 35 36 #
include <G4TScoreNtupleWriter.hh>37 #include <G4RunManagerFactory.hh>38 #include <G4PhysListFactory.hh>
39 #include <G4AnalysisManager.hh>40 #include <G4ScoringManager.hh>41 #include <G4VisExecutive.hh>42 #
include <G4UIExecutive.hh>43 #include <G4UImanager.hh>44 int main(int argc, char** argv) 45 { 46 auto run =
G4RunManagerFactory::CreateRunManager(); 47 G4ScoringManager::GetScoringManager(); // 启用命令式计分器 48
G4TScoreNtupleWriter<G4AnalysisManager>writer; // 启用N元组记录 49 if (run->GetRunManagerType() !=
G4RunManager::sequentialRM) 50 writer.SetNtupleMerging(true); // 合并在多线程中创建的元组 51 // 加载默认物理列表,
或由 $PHYSLIST 指定的物理列表 52 G4PhysListFactory f; run->SetUserInitialization(f.ReferencePhysList()); 53 run->
SetUserInitialization(new Detector); // 加载探测器定义 54 run->SetUserInitialization(new Action); // 加载粒子生成器
55 56 G4UIExecutive* ui = nullptr; // 默认采用批处理模式 57 if (argc==1) ui =new G4UIExecutive(argc, argv); // 交互
模式 58 // 在创建 UI 和启动会话之间启用可视化 59 auto vis = new G4VisExecutive("quiet"); vis->Initialize(); 60 if (ui) { //
启动交互式 UI 会话 61 ui->SessionStart(); delete ui; // 在 UI 会话结束时清理内存 62 } else { // 在批处理模式下运行宏文件
63 G4String cmd ="/control/execute ", macroFile =argv[1]; 64 G4UImanager::GetUIpointer()->
ApplyCommand(cmd +macroFile); 65 } 66 delete vis; 67 delete run; 68 }
```

清单1：通用

, 单文件 Geant4 应用程序模板 , *mingle.cc* , 完整地。

表 1：标有 Git 分支和标签的 Geant4 应用程序开发指导路线图。

| 阶段 | 里程碑 | 分支 | Tag | 重点 |
|---|---|---|---|---|
| 0 | 用户界面 | ui | v0 | 通过文本用户界面/图形用户界面与 Geant4 交互，并浏览宏命令树 |
| 1 | 批处理执行 | 批处理 | v1 | 从交互式探索过渡到自动化批处理执行 |
| 2 | 运行管理器 | run | v2 | 初始化模拟内核并了解模拟生命周期 |
| 3 | 物理列表 | 物理 | v3 | 使用 G4PhysListFactory 选择经过验证的高级物理过程 |
| 4 | 探测器定义 | 探测器 | v4 | 使用简单语法在文本文件中定义探测器几何结构和材料 |
| 5 | 可视化 | vis | v5 | 使用 G4VisExecutive 渲染三维几何体和轨迹 |
| 6 | 粒子生成 | gps | v6 | 使用通用粒子源动态控制粒子生成 |
| 7 | 直方图生成 | 计分器 | v7 | 使用内置命令行网格计分器生成统计分布 |
| 8 | ntuple生成 | 元组 | v8 | 使用 G4AnalysisManager 将逐事件数据保存到元组中 |
| 9 | 磁场 | 场 | v9 | 添加均匀磁场以模拟粒子偏转 |

不仅用于版本控制、代码托管和分发，也可作为面向新手应用开发者的教学工具。

### *2.1. Git* 作为教学工具

教授 Geant4 这类复杂软件框架，主要有两种方法。第一种是拆解一个已完成、功能齐备的应用程序，了解它的工作原理。然而，对于初学者而言，这种逆向工程往往就像试图通过观察一辆组装完成的汽车来了解发动机的工作原理。第二种方法，也是 MinGLE 采用的方法，是逐步完善代码。逐个组件构建应用程序，人们便能了解添加每一行代码的具体必要性。这种自底向上的方法确保人们不会被样板代码淹没，因为每个阶段恰好只引入一个新概念。

为了推动这一循序渐进的学习过程，MinGLE 利用了 Git 版本控制系统的两项核心功能：分支 [8]和标签 [9]，，以构建一条有指导性的开发路线图，如表1所示。MinGLE 中的每个分支都包含一个独特 Geant4 组件或功能的添加，且该组件或功能完全可用并可测试；分支的末端带有标签 v*n*，代表该开发阶段完成的里程碑。这种结构具有以下教学目的：

**概念隔离** 切换到特定分支后，只能看到该里程碑所需的代码，避免受到尚未介绍的概念所带来的“干扰”。

**主动实践** 与静态代码片段不同，分支是一个可实时操作的工作区。你可以修改代码并提交自己的更改，查看它们对模拟产生的影响。

**稳定的检查点** 标签提供了“正确答案”。如果在分支中的实验导致错误，你可以随时将自己的成果与相应标签进行比较，以找到解决方案。

#### *2.1.1.* 用于静态检查的 *Git* 标签

由于 Git 的版本控制特性，使用以下命令获取代码仓库的本地副本后，

```
Git 克隆 https://github.com/jintonic/MinGLE
```

始终可以使用

```
Git 日志
```

以查看代码仓库的完整开发历史。历史中的每个快照都可以通过由一串随机字符组成的提交 ID 获取，例如，

```
Git 检出 b196f6
```

然而，这种原始的按时间顺序排列的历史记录并非线性，杂乱无章，其中包含大量用于调试版、重构的提交，有时还会走入死胡同。浏览这样的历史记录会让新手感到困惑，适得其反。

建议初学者使用 标签 命令，列出经过精心整理且名称明确的开发里程碑：

```
git
标签
v0v1...
```

使用 `show` 命令查看特定里程碑：

```
git show v0:mingle.cc
```

并使用`diff` 命令检查两个里程碑之间的增量更改：

```
Git 差异 v0 v1 MinGLE.cc
```

#### *2.1.2.* 用于动态探索的 *Git* 分支

对于不惧更改代码、喜欢冒险的用户，可以使用以下命令获取所有分支的列表

```
Git 分支 -a* mainremotes/origin/HEAD ->
o
r
i
g
i
n/mainremotes/origin/batch...remotes/origin/main...
```

其中，`*` 符号表示当前本地分支，而`-a` 选项会列出本地分支和远程分支。

刚克隆的代码仓库只有一个本地分支，即`main` 分支。要将工作目录切换到另一个分支（例如 `batch`），可以使用

```
git 切换 批次
```

该命令会从远程仓库拉取该分支，并将工作目录切换到该分支。

如果用户在探索过程中修改了代码，并尝试切换回去，Git 可能会阻止切换，以保护他们的工作。在这种情况下，他们可以使用以下命令丢弃更改，恢复到干净状态

```
git 还原 .
git 切换 主分支
```

或者使用以下命令保存更改后再切换

```
git 暂存
git 切换 main
```

并在之后使用以下命令取回所做的更改

```
git stash pop
```

### *2.2. GitHub* 作为教学工具

通过`git`学习版本控制的基础知识，是每位有志成为开发者的人迈出的重要第一步。不过，不应要求初学者必须精通`git`才能学习 Geant4。将 MinGLE 的 Git 代码仓库托管在 GitHub 上，用户便可以通过 GitHub 网页界面实现同样的目标，无需精通`git`命令。

GitHub 上的 MinGLE 代码仓库，`https://github.com/jintonic/mingle`，列出了代码仓库中的所有文件，用户可以点击任意文件查看其内容。它还会将 README 文件以 Markdown [10] 格式渲染成排版清晰、易于阅读的网页。Markdown 也是一种纯文本格式，在本地计算机上的任何简单文本编辑器中都易于阅读。选择这种格式，是为了确保在线和离线时都能获得最佳呈现效果。

README 文件的内容因分支而异。这种转变摆脱了单体式文档风格，显著提升了代码仓库的教学价值。项目将`main`分支的 README 文件定位为高层次的“教学大纲”，为 MinGLE 的理念提供清晰的概要，并为导航提供核心路线图。与此同时，各个分支的 README 文件则如同专门的“章节”，在用户需要时深入讲解特定概念。同样，示例 Geant4 宏文件*run.mac* 也因分支而异，用户可以点击查看文件内容。这种“即时”学习方式构建了一条逻辑清晰、循序渐进的学习路径，并通过将复杂内容限定在相关开发阶段来降低认知负荷。

可以使用文件列表上方的分支和标签选择器，静态浏览代码仓库的标签和分支。也可以通过以下 URL 直接访问：`https://github.com/jintonic/mingle/tree/v0`其中`v0`可以替换为任意标签或分支名称。

每个代码阶段都可以从以下位置下载为 zip 或 tar.gz 文件，以便检查或测试：`https://github.com/jintonic/mingle/tags`。

GitHub 还提供了对分布式计算的实践性入门。它要求用户将工作内容的同步备份至少保存在两个相互独立的位置：GitHub 服务器和本地计算机，从第一天起就培养稳健的数据管理实践。

最后，对于打算将 MinGLE 用作自己新项目的简洁模板的用户，GitHub 允许他们复刻 MinGLE 代码仓库，从而在 GitHub 上创建属于自己的独立代码仓库。

## 3. 编译与部署

MinGLE 旨在能够在主流操作系统（OS）和环境中运行，以确保初学者能够最大限度地使用它。

该应用附带一个标准的 *CMakeLists.txt* 文件，如清单 2所示。开发者可以使用 CMake，轻松地在三大主流平台 Windows、macOS 和 Linux 上将源代码编译为可执行文件 [11]。

由于构建过程会生成大量临时文件，建议使用单独的`构建`目录，以保持源代码目录整洁。CMake 提供了一种简单的方法，可以使用`-B`选项来实现，该选项用于指定 **B**inary 可执行文件的存放目录：

```
cmake -B 构建
```

请注意，该命令区分大小写。不能用`-b`替代`-B`。

要构建（编译）应用程序，请在 Linux、macOS 和 Windows 上统一运行以下命令：

```
cmake --build build --config 发布版
```

在 Linux 或 macOS 上，也可以使用更短的命令：

```
make -C 构建
```

在 Windows 上，可以在 Git Bash 终端中执行相同的命令 [12]；该终端随 Git for Windows 的标准安装一起提供。对于不熟悉命令行的用户，也可以通过 CMake 和 Visual Studio [13] 的图形用户界面（GUI）完成这些操作。有关具体操作的详细说明，请参阅作者的 YouTube 频道[14]。

选项`--config Release`用于为多配置生成器（例如 Visual Studio、Xcode）指定配置类型，该类型可以是`发布版`、`调试版`、`最小尺寸发布版`、`带调试信息的发布版`或任意自定义类型。对于单配置生成器（例如 Linux 或 macOS 上使用的 Makefile），则不需要此选项。清单 2 中所示的 *CMakeLists.txt* 文件第 9 至 15 行用于将`发布版`指定为默认的构建和配置类型，供不了解`发布版`与`调试版`区别、并且很可能会在 Windows 上的 Visual Studio 中接受此默认设置的用户使用。一般来说，`发布版`类型的可执行文件比`调试版`类型的更精简、运行速度更快。尽管对于 MinGLE 这样小巧的程序而言，这一点并不明显，但对于基于 MinGLE 开发的较大型应用程序，其优势会更加显著。

然而，在 Windows 上的 Git Bash 终端中，*CMakeLists.txt*里的这一默认设置会被`cmake --build`忽略。在这种特殊情况下忽略`--config Release`可能会导致无法.....

```
1 cmake_minimum_required(VERSION 3.16) # CMake 不加这一项就会退出 2 3 project(mingle) # 项目名称 4 5 #
http://geant4-userdoc.web.cern.ch/geant4-userdoc/UsersGuides/InstallationGuide/html/buildtools.html 6 find_package(Geant4
REQUIRED ui_all vis_all) # 查找 Geant4Config.cmake 7 include(${Geant4_USE_FILE}) # 没有这一项，Linux 无法找到 Geant4 头文件 8
9 if(WIN32) # 为多配置生成器设置配置类型，例如 Visual Studio、Xcode 10 set(CMAKE_CONFIGURATION_TYPES Release CACHE
STRING "Release;Debug;..." FORCE) 11 else() # 为单配置生成器设置默认构建类型，例如 Makefile 12 if(NOT CMAKE_BUILD_TYPE) # 如果
用户未定义 13 set(CMAKE_BUILD_TYPE Release CACHE STRING "Release;Debug;..." FORCE) 14 endif() 15 endif() # 发布版类型的可执行
文件比调试版更精简、速度更快 16 17 if(CMAKE_INSTALL_PREFIX_INITIALIZED_TO_DEFAULT) # 如果用户未定义 18 set(CMAKE_
INSTALL_PREFIX "${Geant4_DIR}/../../.." CACHE PATH "安装前缀" FORCE) 19 endif() # 设置默认安装前缀 20 21 if(APPLE) #
https://gitlab.kitware.com/cmake/community/-/wikis/doc/cmake/RPATH-handling 22 set(CMAKE_INSTALL_RPATH "${Geant4_
DIR}/../..") 23 endif() # 在 Mac 上的可执行文件中设置运行时搜索路径（必须在 add_executable(...) 之前） 24 25 add_executable($
{PROJECT_NAME} ${PROJECT_NAME}.cc) 26 target_link_libraries(${PROJECT_NAME} ${Geant4_LIBRARIES}) 27 install(TARGETS $
{PROJECT_NAME}) # 启用 cmake --install
```

清单 2：*CMakeLists .txt* 确保生成发布版可执行文件，并将其安装到 Geant4 的 bin/ 目录 .

预期行为。建议 Windows 终端用户始终指定它。

要安装该应用程序，请在 Linux、macOS 和 Windows 上统一运行以下命令：

```
cmake --install 构建
```

或者通过 Visual Studio 图形用户界面构建 `INSTALL` 目标。

列表 2 中所示的 *CMakeLists.txt* 第 17 至 19 行指示该命令将生成的 mingle 可执行文件复制到构建过程中由 CMake 检测到的本地 Geant4 安装的`bin`目录中。由于成功完成 Geant4 安装后，该目录本就必须位于用户的 PATH 环境变量中，因此这一设置使 mingle 可以从任意位置全局使用，而无需额外配置环境。于是，用户只需输入

```
mingle
```

即可在终端中，甚至在 Windows 文件浏览器的地址栏中启动该应用程序。

MinGLE 的设计注重简洁，同时还通过采用 JSON 编译数据库标准支持高级开发工作流 [15]。设置环境变量

```
CMAKE 导出编译命令=开启_ _ _
```

在配置步骤中，CMake 会自动在构建目录中生成*compile_commands.json* 文件 [11]。

该文件充当构建系统与现代集成开发环境（IDE）或文本编辑器之间的桥梁，例如 Visual Studio Code [16]、、Neovim [17], 或 CLion[18], 等。这些编辑器会使用编译数据库来解析 Geant4 专用头文件路径和编译器标志。对于开发者而言，这使 IntelliSense 功能得以启用，包括实时语法高亮、精准的代码导航（例如“转到定义”）以及针对复杂 Geant4 类的自动补全，无需手动配置特定于编辑器的包含路径。将此配置保存在用户的环境中（例如 *~/.zshenv* 或 *~/.bashrc*），而不是将其硬编码到*CMakeLists.txt*中，项目便能在保持核心精简的同时，为严肃的应用开发提供现代化环境。

为应对新用户可能难以处理本地编译依赖项这一常见问题（例如 Geant4 安装、系统库），我们通过 Docker 镜像 [19]托管在 Docker Hub [20]上，提供预编译的可执行文件以及完整的 Geant4 库。这样，即使用户无法自行编译，也能立即体验应用程序的全部功能。使用 Docker 容器还可提供开箱即用的体验，使用户免受特定操作系统的编译问题困扰，从而专注于学习 Geant4 应用程序的开发和运行时配置。

## 4. 开发路线图

开发路线图围绕十个关键阶段展开，每个阶段都以一个描述性的 Git 分支和标签名称标记，如表 1所示。接下来的小节将逐步梳理这一开发历程，展示每个阶段所需的最小代码增量。

### 4.1. *Geant4* 用户界面

此阶段的 *mingle.cc*仅包含 6 行 C++ 代码，核心是 `G4UIExecutive`类 [21]:

```
#include <G4UIExecutive.hh>2 整型主函数(整型 参数计数, 字符**参数向量)3 {4 G4UIExecutive 用户界面 ( 参数计数, 参数向量);5 用户界面.启动会话();6 }
```

由于缺少探测器和物理组件等，MinGLE 在此阶段尚不能作为模拟程序运行。不过，它完全可以执行，并为学习 Geant4 交互会话和宏命令结构提供了多个即时、实践性的机会。

Geant4 宏命令是基于文本的指令，允许用户在运行时控制模拟的几乎所有方面，而无需编写或编译额外的 C++ 代码。这种机制是 MinGLE 在保持精简 C++核心的同时实现高度可配置性的重要途径之一。

这些命令以分层树结构组织，类似于 Unix 文件系统，相关功能归入诸如 `/control/`、`/units/`、`/gui/` 等目录中。在此阶段，这三个目录（或命令集）是仅有的可用项。随着程序不断发展，它们也会随之扩展。

默认情况下，运行 `mingle` 会打开一个 Geant4 图形用户界面窗口（如果用户本地安装的 Geant4 启用了该功能），以可视化方式与环境交互。图 1 展示了 Mac 上的图形用户界面屏幕截图1。

用户可以在执行前设置环境变量 `G4UI_USE_TCSH` ，轻松切换到文本用户界面（TUI）。在生成的会话中，用户可以使用类似文件系统的 `ls` 和 `cd` 命令，以交互方式浏览分层的 Geant4 宏命令树，如下所示:

```
G4UI_USE_TCSH=1 mingle 可用的 UI 会话类型: [ Qt、tcsh、csh ] 预初始化> ls命令目录路径: / 子目录: /control/ 用户界面控制命令。/units/ 可用单位。命令: 预初始化> cd /control/预初始化> ls命令目录路径: /control/说明: 用户界面控制命令。子目录: /control/cout/ 控制本地线程的 cout/cerr 命令: macroPath * 设置宏搜索路径..... execute * 执行宏文件。.....
```

在文本用户界面模式下无法使用，因此不存在`/gui/` 。

请注意，在 Geant4 中，模拟处于不同状态，这些状态决定了哪些命令可用，以及工具包当前正在执行什么操作。终端中显示的提示符（例如，`预初始化>`）会动态变化，以反映这些状态。预初始化 是第一个状态。其他状态将在后续章节中介绍。

用户还可以练习将宏命令组合到文件中，例如*run.mac*:

```
# 打印重要的环境变量/control/getEnv PATH/control/getEnv GEANT4_DATA_DIR/control/listAlias
```

图 3：适用于 v0 的 Geant4 宏文件*run.mac*。

随后可以在 Geant4 交互会话中使用`execute` 命令执行此文件:

```
预初始化> /control/execute run.mac
```

对于更习惯使用图形用户界面的用户，可以点击 Geant4 图形用户界面窗口左上角的文件夹图标，加载辅助宏文件 *menu.mac*，利用`/gui/` 目录中的宏命令，将常用命令添加到图形用户界面的菜单栏中。用户只需从菜单中选择一项，即可执行相应的宏命令。

菜单会随着程序的演进而扩展。在此阶段，它仅包含四个顶层项目：“执行”、“详细程度”、“窗口”和“帮助”。在“帮助”中，用户可以选择“在屏幕上打印宏命令”或“生成 HTML 格式的手册 ”，查看此阶段可用的宏命令列表及其说明。好奇的用户可以打开 *menu.mac* ，查看与菜单项对应的宏命令。完整的 *menu.mac*如清单 4所示。

### 4.2. 批处理执行

`ui` 分支引入了交互式 UI 会话。`batch` 分支侧重于一个基本概念：通过检查命令行参数，区分交互式执行与非交互式（批处理）执行。

用户可以通过查看两个分支之间 *mingle.cc* 的差异，了解实现这两种功能所需的修改:

```
git diff 界面批处理 MinGLE.cc
```

输出突出显示了对 主 函数的修改:

```
diff --git a/mingle.cc b/mingle.cc ...+#include <G4UImanager.hh>int main(int argc,char** argv) { - G4UIExecutive ui(argc, argv); - ui.SessionStart();+ if (argc == 1) { // 启动交互式用户界面会话+ G4UIExecutive ui(argc, argv); + ui.SessionStart();+ } else { // 在批处理模式下运行宏文件+ G4String cmd="/control/execute ", macroFile= argv[1];+ G4UImanager::GetUIpointer() ->ApplyCommand(cmd + macroFile);+ } }
```

程序会检查参数的数量（`argc`）以确定执行模式。如果只有一个参数，即程序名称，程序便会进入交互模式，启动 Geant4 会话并等待用户输入。如果有两个

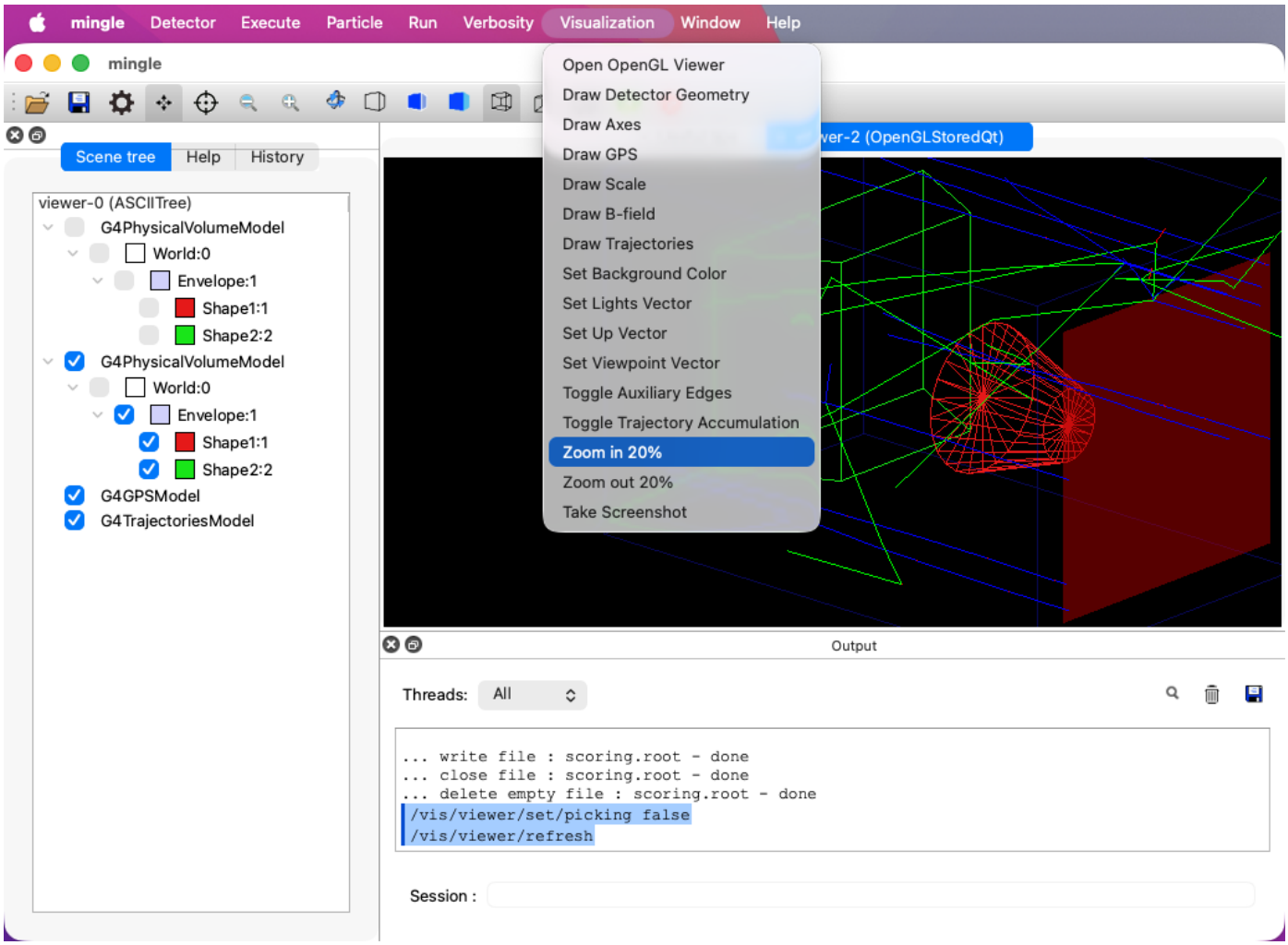



图 1：使用自定义菜单、可视化探测器几何结构并显示累积粒子轨迹的 Geant4 图形用户界面屏幕截图 es.

个参数，程序便会进入批处理模式，执行由第二个参数指定的宏文件：

```
# 程序名称本身也计为一个参数mingle run.mac 在批处理模式下运行
```

该 `ui` 版本要求用户进入交互式会话才能执行宏文件。相比之下，批处理 版本使用 `G4UImanager` 类 [22] 在 C++ 中实现这一功能。下面的 *run.mac* 使用 `/control/` 目录中的宏命令，根据执行模式更改输出消息：

```
/control/doifBatch /control/echo 在批处理模式下运行/control/doifInteractive /control/echo 在交互模式下
```

其中，`/control/echo` 用于在屏幕上打印消息。

### *4.3.* 运行管理

运行分支引入了一个负责管理模拟的核心组件：运行管理器。用户可以通过 Git 差异命令查看初始化模拟内核所需的最少 C++新增内容：

```
Git 差异 批量运行 MinGLE.cc
```

输出突出了对 主函数的修改：

```
   ...+#include  <G4RunManagerFactory.hh> ...
intmain(intargc,char** argv){+    auto运行 = G4RunManagerFactory::CreateRunManager();    if(argc==1){...}
+    delete运行;}
```

代码使用现代的`G4RunManagerFactory` 类[23]来创建运行管理器。这种方法展示了 Geant4 的一个关键现代概念：通过 `G4RUN_MANAGER_TYPE` 环境变量，动态切换默认的多线程（MT）执行模式和串行模式。这使用户能够轻松调试并比较多线程模式的性能。以下终端输出展示了这种无缝切换：

```
   mingleGeant4 版本名称：geant4-... [MT] (...)
<< 处于多线程模式 >>...
```

列表 5：默认（多线程）执行。

```
# 在屏幕上打印每条宏命令/control/verbose


# 清除现有菜单
/gui/clearMenu

/gui/addMenudet 探测器
/gui/addButton det "检查体积重叠（在 /run/initialize 后）" "/geometry/test/run"/gui/addButton
det "为体积创建计分器" "/score/create/realWorldLogVol"/gui/addButton det "记录剂量沉积" "/
score/quantity/doseDeposit"/gui/addButton det "记录能量沉积" "/score/quantity/energyDeposit"/
gui/addButton det "关闭计分器" "/score/close"/gui/addButton det "将品质转储到文件" "/
score/dumpQuantityToFile"/gui/addButton det "绘制计分投影" "/score/drawProjection"/
gui/addButton det "设置均匀磁场" "/globalField/setValue"

/gui/addMenu exe 执行
/gui/addButton exe "系统命令" "/gui/system"/gui/addButton exe "宏
文件" "/control/execute"

/gui/addMenu 运行 粒子
/gui/addButton 运行 "设置粒子" "/gps/particle"/gui/addButton 运
行 "设置能量" "/gps/energy"/gui/addButton 运行 "设置方向" "/
gps/direction"/gui/addButton 运行 "设置位置" "/gps/pos/centre"


/gui/addMenu 运行 运行
/gui/addButton 运行 "初始化" "/run/initialize"/gui/addButton 运行 "设置线
程数" "/run/numberOfThreads"/gui/addButton 运行 "设置事件数" "/run/beamOn"/
gui/addButton运行 "10 个事件" "/run/beamOn 10"/gui/addButton运行 "100 个事
件" "/run/beamOn 100"/gui/addButton 运行 "1000 个事件" "/run/beamOn 1000"


/gui/addMenu 详细程度 详细程度
/gui/addButton verb "禁用电磁过程详细程度" "/process/em/verbose 0"/gui/addButton verb "禁用
强子过程详细程度" "/process/had/verbose 0"/gui/addButton verb "启用磁场详细程度" "/
globalField/verbose 1"/gui/addButton verb "启用运行详细程度" "/run/verbose 2"/
gui/addButton verb "设置宏命令详细程度" "/control/verbose"/gui/addButtonverb "设置径迹跟踪详细
程度" "/tracking/verbose"


/gui/addMenu 可视化 可视化
5 /gui/addButton vis "打开 OpenGL 查看器" "/vis/open OGL" 6 /gui/addButton vis "绘制探测器几何结构" "/
vis/drawVolume World" 7 /gui/addButton vis "绘制坐标轴" "/vis/scene/add/axes" 8 /gui/addButton vis "绘制
GPS" "/vis/scene/add/gps" 9 /gui/addButton vis "绘制比例尺" "/vis/scene/add/scale" 0 /gui/addButton vis "绘
制磁场" "/vis/scene/add/magneticField" 1 /gui/addButton vis "绘制轨迹" "/vis/scene/add/trajectories rich" 2 /
gui/addButton vis "设置背景颜色" "/vis/viewer/set/background" 3 /gui/addButton vis "设置光照向量" "/
vis/viewer/set/lightsVector" 4 /gui/addButton vis "设置向上向量" "/vis/viewer/set/upVector" 5 /
gui/addButton vis "设置视点向量" "/vis/viewer/set/viewpointVector" 6 /gui/addButton vis "切换辅助边" "/
vis/viewer/set/auxiliaryEdge" 7 /gui/addButton vis "切换轨迹累积" "/vis/scene/endOfEventAction" 8 /
gui/addButton vis "放大 20%" "/vis/viewer/zoom 1.2" 9 /gui/addButton vis "缩小 20%" "/vis/viewer/zoom
0.8" 0 /gui/addButton vis "截取屏幕截图" "/vis/ogl/export detector.png"
5
5
6
61
62 /gui/addMenu window 窗口
63 /gui/addButton window 关闭 退出
64
65 /gui/addMenu帮助 帮助
66 /gui/addButton help "在屏幕上打印宏命令" "/control/manual /" 67 /gui/addButton help "以 HTML 格式生成帮助文
档" "/control/createHTML" 68 /gui/addButton help "列出分析对象（在 /run/initialize 之后）" "/analysis/list 0" 69 /
gui/addButton help "列出材料" "/material/nist/listMaterials all" 70 /gui/addButton help "列出粒子" "/particle/list all"
71 /gui/addButton help " 列出过程（在 /run/initialize 之后） " "/process/list all" 72 /gui/addButton help " 列出单位 " "/
units/list" 73 /gui/addButton help "列出计分器" "/score/list" 74 /gui/addButton help "列出计分器颜色映射" "/
score/colorMap/listScoreColorMaps"
```

清单 4：用于自定义 Geant4 图形用户界面菜单栏的完整*menu.mac* 文件。

```
G4RUN_MANAGER_TYPE=串行 mingle
环境变量“G4RUNMANAGER TYPE”_ _已启用，
值为 ==Serial。正在覆盖 G4RunManager 类型.....

Geant4 版本名称：geant4-... [串行] (...) ...
```

清单 6：串行执行（由环境变量覆盖）。

在main 函数末尾添加delete run;，有助于培养正确管理 C++ 内存的责任意识，确保模拟内核分配的资源在程序终止时得到正确释放。对于初学者而言，这是一个新概念；他们可能熟悉 Python 等语言，而在这些语言中，内存管理是自动处理的。

实例化运行管理器后，首次向用户注册了一套全面的宏命令：

```
命令目录路径：/ 子目录：/control/ UI 控制命令。/units/ 可用
单位。/particle/ 粒子控制命令。/tracking/
TrackingManager 等...../geometry/ 几何控制命令。/process/
过程表控制命令。/event/ EventManager 控制命令。/cuts/
G4VUserPhysicsList 的命令。/run/ 运行控制命令。/random/
随机数状态控制命令。
```

这显著扩展了交互控制能力，超出了此前见到的最简/control/、/units/和/gui/ 目录。不过，其中许多只是占位符，只有在稍后实现相应组件后才会派上用场。

在这个阶段，少数宏命令确实开始发挥作用。例如，/units/list 现在可以打印 Geant4 使用的单位列表。它作为一个选项被添加到*menu.mac* 的“帮助”菜单中。更多用于控制详细程度的命令也被添加到“详细程度”菜单中。/random/setSeeds被添加到*run.mac*，以演示如何使用整数种子流初始化随机数生成器。

### *4.4.* 物理列表

物理列表定义了哪些过程会发生、特定粒子以何种频率发生这些过程，以及相应的能量范围。使用G4PhysListFactory类 [24]，，physics 分支使初学者能够以最简单、最受推荐的方式选择物理列表。只需在run分支的基础上添加两行代码即可实现：清单1中的第 38 行和第 52 行。

G4PhysListFactory 使初学者无需低级地“组装”单个粒子和过程，即可获得一套有意义且经过验证的参考物理列表。用户可以使用 PHYSLIST 环境变量在运行时切换物理列表，如下所示：

```
物理列表=FTFP_BERT_EMV mingle

... G4PhysListFactory::GetReferencePhysList<FTFP_BERT_
EMV> EMoption= 1<<<Geant4 物理列表模拟引擎：FTFP_BERT

G4VModularPhysicsList::ReplacePhysics：G4EmStandard 类
型为 2，已替换为 G4EmStandard_opt1
```

定义物理列表后，应用程序的宏命令得到进一步扩展，为用户新增了两组功能：1. /material/ 用于查看内置材料；2. /physics_lists/用于切换特定的物理过程。

/particle/list 现在确实可以用于列出 Geant4 中所有可用的粒子，并与*menu.mac* 中的/material/nist/listMaterials 一起作为“帮助”菜单中的项目。此外，/process/ 和/particle/目录中还提供了更多子命令。

### *4.5.* 探测器定义

该探测器分支通过定义Detector 类及其在清单1第 53 行中的实例化，引入了一个必需的用户初始化组件：探测器定义。然而，不同于在 C++中硬编码体积的传统方法，MinGLE 采用了 Geant4 文本几何格式 [25]，——这是 Geant4 自 9.2 版本起提供的标准功能，通过 G4tgbVolumeMgr类 [26]实现。

例如，可以在名为 *detector.tg* 的文本文件中，仅用一行文本定义一个充满空气的 $10\times10\times10\ \mathrm{m}^3$ 立方体实验大厅：

```
:volu 大厅 BOX 5*m 5*m 5*m G4 空气_
```

相比之下，在 C++ 中定义同一个“hall”需要经过多个步骤，实例化至少四个不同的对象：实体（BOX）、材料（G4_AIR）、逻辑体积和物理体积（hall），通常需要五行以上代码。:volu 语法将这种 C++ 层级结构简化为一行易读的文本，大幅减少了初次使用该工具包的用户所面对的“语法噪音”。

从外部文本文件加载探测器定义，可使 C++ 代码的长度和复杂度保持不变，无论探测器有多复杂。无论是模拟一个简单的盒体，还是包含数千个体积的复杂探测器，C++代码始终保持为简洁的 4 行接口（清单1第 8 至 11 行）。

此外，通过将探测器定义与 C++源代码分离，MinGLE 成为了通用引擎。用户可以修改探测器的尺寸或材料，并立即重新运行模拟。这样便省去了典型 C++ 程序中的编译-链接-运行周期，从而能够快速试验几何结构对物理结果的影响。

为了清楚展示上述所有优点，Geant4 基础示例 B1 [27] 中分布在两个文件里的 100 多行 C++ 代码，在 *detector.tg* 中精简为仅 8 行：

```
:
volu World BOX 120 120 180G4 空气_:volu Envelope BOX 100
100150 G4_水 :volu Shape1 CONS 0 20 0 4030 0 360 G4_A-150_组织 :
volu Shape2 TRD 60 60 50 80 30 G4_骨_致密_ICRU :rotm r000 0 0 0 :
place Envelope 1 World r000 0 0 0 :place Shape1 1 Envelope r000 0
20 -70 :place Shape2 2 Envelope r000 0 -10 70
```

图 7：在*detector.tg*中重新定义的 Geant4 基础示例 B1 探测器。

其中定义了一个由 A-150组织等效塑料制成的切割圆锥体（红色）和一个由致密骨制成的梯形体（绿色），并将它们放置在充水箱（蓝色）中，如图所示1。

要使用 *detector.tg* 中定义的几何，必须将该文件放置在执行`mingle` 的目录中。否则，Geant4 将提示该文件不存在。

请注意，外部文件名 *detector.tg* 是硬编码的，这样可以使模板保持简洁，并避免实现用于加载任意文件名的自定义宏命令。对于希望加载其他名称文件的用户，可以在 *detector.tg* 中使用内置的 `#include` 指令，指向其他目录中的另一个文件：

```
#include /path/to/my/favorite/experiment.t g
```

虽然 GDML [28] 是另一种常见的外部几何标准，但它需要 Xerces-C库 [29]；MinGLE 使用文本几何而不是 GDML，因为文本几何能够提供相同的解耦优势，同时不会增加额外依赖，因此对初学者更加友好。

定义好探测器后，终于可以使用以下命令初始化运行，而不会导致程序崩溃：

```
/run/initialize
```

它将加载探测器几何结构并初始化物理过程。之后，可以使用 `/geometry/test/run`检查探测器体积之间是否存在重叠，也可以使用 `/process/list` 列出所有已初始化的物理过程，如下方包含的 *run.mac*所示：

```
1  # 在屏幕上打印每条宏命令2  /control/verbose34  #
避免线程输出交错5  /control/cout/useBuffer67  # 禁用大量
输出8  /process/em/verbose09  /
process/had/verbose01011 # 详细显示运行的执行过程12  /
run/verbose213  # 加载探测器定义和物理过程14  /
run/initialize1516  # 检查探测器体积之间是否存在重叠17  /
geometry/test/run1819  # 打印物理过程20  /process/list
```

清单8： *run.mac*在 MinGLE v4 中。

默认情况下，`/run/initialize` 会在屏幕上输出大量信息，可能会让初学者不知所措。为避免这种情况，*run.mac* 中的第 8 行和第 9 行用于禁用电磁和强子物理过程的详细输出。另一方面，第 12 行用于在屏幕上详细打印运行的执行过程：

```
...# 详细显示运行的执行过程/run/verbose 2# 加载探测器
定义和物理过程/run/initializeuserDetector->
Construct() 开始。
World 已注册到默认区域。
physicsList->构造()开始。
...
physicsList->检查粒子列表()开始。
physicsList->设置截断值()开始。
# 检查探测器体积之间的重叠
/几何/测试/运行
正在运行几何重叠检查……正在检查体积 Envelope:1 (G4Box) 的重叠……正
常！正在检查体积 Shape1:1 (G4Cons) 的重叠……正常！正在检查体积
Shape2:2 (G4Trd) 的重叠……正常！几何重叠检查已完成！

# 打印物理过程
/process/list
    输运、伽马通用过程、多重散射、电子电离……G4内核已进入退出状态。
用户探测器构造已删除 0xaa6e06de0 用户物理列表已删除 0x109c777d0 用户
动作初始化已删除 0x0 用户工作线程初始化已删除 0x0 用户工作线程初始化已
删除 0x0 用户运行动作已删除。用户初级粒子生成器已删除。运行管理器正在
删除运行管理器内核。G4敏感探测器管理器已删除。EventManager 已删除。
单位表已清除。输运管理器已删除。共清理导航历史集合：2。G4随机数生成
辅助对象已删除。……

    G4分配器对象已删除。用户界面管理器已删除。状
态管理器已删除。运行管理器内核已删除。再见 :) 运行管
理器已删除。
```

这清楚地展示了每个组件何时被构造以及随后被删除。

默认情况下，Geant4 以多线程模式运行。每个线程都会打印自己的输出。为避免不同线程的输出交错，第5 行的 *run.mac*用于将每个线程的输出汇总到一个缓冲区中，然后一次性打印一个缓冲区。不过，这仍不理想，因为每个线程的缓冲区中都有大量重复信息。要获得最整洁的输出，只能以串行模式运行 `mingle`，如条目6所示。

*menu.mac* 在此阶段进一步扩展，使用户能够通过 Geant4 图形用户界面执行 *run.mac* 中的各条命令。

调用 `/run/initialize` 后，会出现新的宏命令集 `/physics_engine/` 。用户可以通过它设置中子跟踪的能量和时间限制。

在文本用户界面中，调用 `/run/initialize` 前后显示的不同提示符会表明 Geant4 状态的变化：

```
    G4UI_USE_TCSH=1
mingle...预初始化> /
run/initialize...空闲>
```

空闲 表示探测器几何、物理过程等基本组件已加载到内存中，Geant4 已准备好运行模拟。

### *4.6.* 可视化

该 可视化 分支可在此阶段显示探测器几何结构，并在后续阶段显示粒子轨迹。这是通过 `G4VisExecutive` 类 [30]实现的，如清单1第 41 行和第 59 行所示。

使用“quiet”参数初始化 `G4VisExecutive`，以抑制可能令初学者困惑的初始详细输出。不过，可以按需使用`/vis/list`打印其详细输出；该命令已添加到*menu.mac*中的“帮助”菜单。

`/vis/list` 输出的一部分是用户特定 Geant4 安装中可用的可视化驱动程序列表：

```
已注册的图形系统包括：
  ASCIITree (ATree)
  DAWNFILE (DAWNFILE)
  RayTracer (RayTracer)
  VRML2FILE (VRML2FILE)
  ...
  TOOLSSG_OFFSCREEN（TSG_OFFSCREEN ， TSG_FILE）
  OpenGLImmediateQt（OGLIQt、OGLI）
  ...
```

图 9：Geant4 中常见的可视化驱动程序。

基于 OpenGL 的可视化如图1所示。

ASCIITree 并不真正进行可视化，而是在终端中打印探测器几何结构的层次结构，类似于图1左侧面板所示。

清单 9 展示了其他可视化驱动程序的用法*run.mac*。它们都会生成一个输出文件，该文件可以是普通图像文件，也可以是可在专用查看器中打开的文件。

典型的工作流程是先使用图 1所示的 Geant4 图形用户界面，通过拖动鼠标和缩放微调探测器几何结构的可视化效果，然后使用其他驱动程序相应地生成图像文件。

关键在于，`G4UIExecutive` 的实例化方式已从栈对象重构为指针 [31]。这样一来，可视化管理器便可在创建 UI 执行器之后、但在会话开始之前完成初始化。自 Geant4 11 版起，必须采用这一特定的初始化顺序，才能启用自动选择可视化驱动程序的功能；Geant4 基础示例 B1 [27] 中的*vis.mac*文件展示了这一用法。

在此阶段，*menu.mac* 新增了一个“可视化”菜单，方便用户通过点击访问常用的可视化命令，如图 1的下拉菜单所示。

探测器体积的颜色和透明度可通过在图形用户界面中单击鼠标、使用宏命令，或将内容追加到文本几何定义文件*detector.tg*中进行调整：

```
  : 可见World 关闭: 颜色Envelope
0.1 0.1 0.90.2: 颜色Shape10.9 0.1
0.1: 颜色Shape20.1 0.9 0.1
```

前三个数字是 RGB 颜色值。最后一个数字是透明度值。如果未指定，默认值为 1.0（完全不透明）。

调用`/run/initialize`后，还会出现一组新的宏命令`/param/`，用于快速模拟（参数化）控制；这一主题超出了 MinGLE 的讨论范围。

### *4.7.* 粒子生成

该GPS 分支实现了生成器类（清单 1第 20–28 行），该类使用 Geant4 组件 `G4GeneralParticleSource` （GPS） [32]生成初级粒子以启动模拟。它还实现了动作类（清单 1第 31–35 行），用于定义生成粒子的动作，并在`main` 函数（清单1第 54 行）中使用。

GPS 允许用户完全通过宏命令 [33], 定义初级粒子的属性，例如类型、能量、位置和角分布，从而无需在 C++ 中硬编码特定的粒子枪。与标准 `G4ParticleGun` 相比，`G4GeneralParticleSource` 类灵活且功能强大得多，因此 MinGLE 默认使用该类作为粒子生成器。

在 *menu.mac* 中，新增了一个“粒子”菜单，方便通过图形用户界面配置常用的 GPS 设置。“运行”菜单中也新增了一些选项，用于设置模拟的事件数。

*run.mac* 在此版本中模仿了 Geant4 基础示例 B 1 [27],，演示如何向几何体发射 210 MeV 质子，并逐步在屏幕上打印模拟步骤。运行后，当前目录中将生成两个 PNG 文件`mingle run.mac`，一个显示几何体，另一个显示几何体 + 粒子轨迹，类似于图1所示。

### *4.8.* 直方图生成

该计分器分支会激活 Geant4 内置组件 `G4ScoringManager` [34]（见清单1第 47 行）。用户可在探测器中的任意体积上定义计分网格（三维网格），以记录能量沉积、剂量或通量等物理量，而无需修改 C++代码。

本阶段演示了*run.mac* 中的两种网格：一种与*detector.tg*中定义的*Shape1*完全相同，用于记录穿过它的质子数量并保存到 CSV 文件中；另一种是 $25 \times 25 \times 1$ 网格，用于记录*Shape2* 后方的剂量分布，并保存为 PNG 文件，如图 2所示。直方图中正对着*Shape2*的中间灰色区域表明，从右侧红色方块射出的许多粒子朝*Shape2* 飞去时被*Shape2*吸收了。

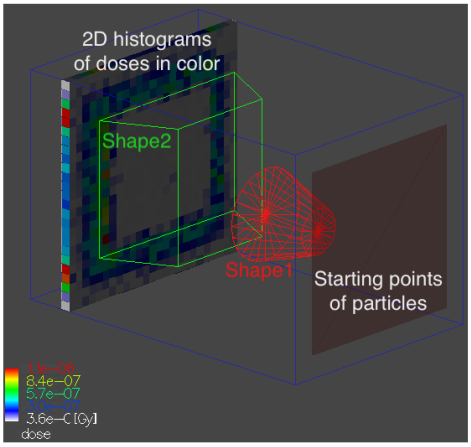



图 2：计分管理器记录的 *Shape2* 后方剂量分布，以彩色二维直方图显示。

在 *menu.mac* 中，“探测器”菜单已展开，用户可以通过图形用户界面将计分器附加到特定体积。

### *4.9.* 元组生成

虽然直方图便于快速检查，但详细分析通常需要逐事件数据。元组分支用于引入`G4TScoreNtupleWriter` [35]（位于清单第 48 行）1。该类会自动将计分管理器记录的数据保存到元组中， [36] 格式为 ROOT TTree [37]。

对于多线程运行，写入器会将不同工作线程中的元组合并到一个输出文件*scoring.root* 中（清单第 49–50 行1）。随后可使用 ROOT 框架 [38]或 Uproot 等 Python 库 [39] 对该输出进行分析。

以下 *run.mac*用于演示如何逐事件记录*Shape1*中的能量沉积：

```
/run/initialize # 使用 60 千电子伏伽马射线检查能谱
/gps/particle gamma
/gps/energy 60 keV
/gps/direction 0 0 1
/gps/pos/centre 0 2 -11 cm # 记录 Shape1 中的能量沉积
/score/create/realWorldLogVol Shape1 1
/score/quantity/energyDeposit e1
/score/close
/run/printProgress 1000
/run/beamOn 10000
```

使用 60 keV伽马射线的全吸收峰，而非高能质子，来检查能量沉积是否被正确记录。第 10 行中的最后一个数字表示 *Shape1* 在探测器层级结构中的深度，而不是其复制编号。

元组的名称通常为`mesh_quantity`，其中`mesh` 是计分网格的名称，本例中为*Shape1*；`quantity` 是要记录的量的名称，本例中`e`表示能量沉积。元组中逐事件记录能量沉积的分支名称为`Shape1_e_score`。所有这些都可以通过以下 ROOT 命令（`.ls` 和 `TTree::Show()`）进行检查：

```
root scoring.root root[0] .ls TFile** scoring.root TFile* scoring.root KEY:TTree Shape1_e;1 Shape1_e root[1] Shape1_e->Show(0) ======> 事件:0 Shape1_e_eventId = 1 Shape1_e_cell = 0 Shape1_e_score = 0.0239774 root[3] Shape1_e->Draw("Shape1_e_score")
```

最后一条命令将逐事件填充一个直方图，其中记录*Shape1* 中的能量沉积，展示使用 ROOT 框架分析元组的有效性。

提供了一个 Python 脚本 *drawE.py* ，使用 Uproot 绘制相同的分布 [39] 和 Matplotlib [40] ，供不熟悉 ROOT 的用户使用。结果如图 3所示。

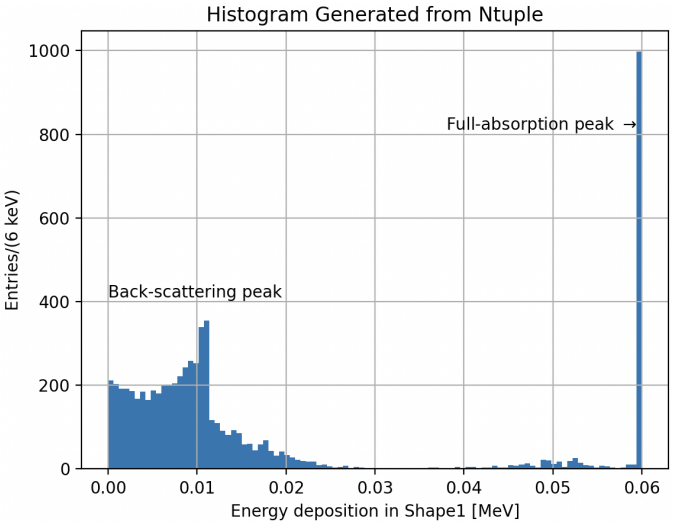



图 3：60 keV伽马射线在*Shape1*中的能量沉积。

由于 `G4TScoreNtupleWriter` 类继承自 `G4AnalysisManager` [41]，，在执行 `/run/initialize` 后，Geant4用户界面中会出现一个新的宏命令目录 `/analysis/` 。

`G4TScoreHistFiller` 类 [42] 也继承自`G4AnalysisManager` 类。它可用于在一次运行期间填充直方图，但由于运行结束后可以使用元组创建具有自定义分箱的直方图，因此选择 `G4TScoreNtupleWriter` 而不是`G4TScoreHistFiller`。

*menu.mac*经过扩展，允许用户使用 Geant4 图形用户界面选择一个体积，并记录其中的能量沉积。

### *4.10.* 磁场设置

最后一个阶段，即`field` 分支，增加了在整个探测器中设置均匀磁场的功能。

`G4GlobalMagFieldMessenger` [43], 在清单1第 12–14 行内注册后，会在目录`/globalField/`下提供一组宏命令，供用户定义均匀磁场矢量：

```
/globalField/setValue 0 0 1 特斯拉
```

*run.mac* 经过精细调整，用于展示 100 MeV$\mu^-$粒子在充水盒（黑色）中的磁场内的轨迹（红色），如图所示4。

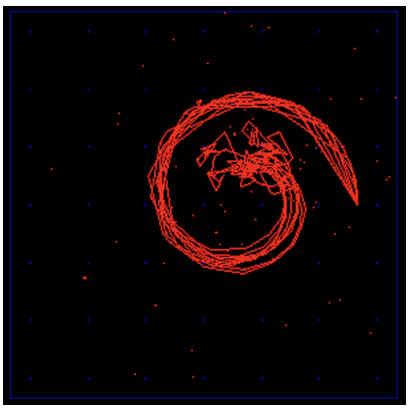


图4：带电粒子在均匀磁场中弯曲的轨迹。

### *4.11.* 主分支

主分支是代码仓库最终可用于生产的状态。虽然核心应用代码 *mingle.cc* 和*menu.mac*与场分支（代表增量开发的最终阶段）中的代码相同，但辅助文件（例如 *run.mac* 和 *drawE.py* ）有所不同，因为这些文件是为更通用的演示量身定制的，而不是专注于介绍某项特定功能。

MinGLE 仓库中的文件在各个分支间采用了结构化的分配策略：

- *CMakeLists.txt* 和 *.gitignore* 在所有分支中完全相同，以确保构建体验一致。
- *mingle.cc*、*detector.tg* 和 *menu.mac* 逐步扩展，每个开发阶段只添加实现新功能所必需的代码行。
- *README.md* （“说明”）和 *run.mac* （“练习”）因分支而异，从而为每个里程碑提供本地化的文档和测试命令。
- MinGLE 标志（*logo.png*）和项目 *LICENSE* 等某些项目级文件仅存在于 main 分支中，以尽可能保持面向学习的分支简洁。

## 5. 讨论

官方Geant4 用户指南包含一个“入门”章节 [44] ，通过一系列 C++ 代码片段向新手开发者介绍重要的 Geant4 组件。这些代码片段对于理解各个类的具体语法和用途非常有价值。然而，它们往往是孤立呈现的，初学者因此无法获得一个完整、可立即编译并进行实验的程序。

MinGLE 有效地填补了这一空白。MinGLE 不提供需要手动拼装的静态代码片段，而是从第一阶段（交互式用户界面）开始，就提供完整且可运行的模拟内核。随着用户沿着路线图逐步学习，他们不仅是在阅读 G4GeneralParticleSource 或 G4tgbVolumeMgr 这类类，还能在经过验证并通过可编译性测试的环境中观察它们实际运行。这种变革性的方式将学习体验从被动阅读代码转变为主动构建应用，为官方文档中的基础组件与完整的工具包实现之间搭起了至关重要的“缺失环节”。

## 6. 结论

MinGLE 为 Geant4 应用程序开发提供了一种现代、极简的范式。通过去除样板代码并运用当代工具包的功能，它将模拟的功能核心精简至绝对必要的部分，同时不牺牲灵活性或性能。其独特的、基于 Git 的教学结构，通过提供一条从简单 UI 到功能丰富应用程序的连续且可编译路径，弥补了官方文档中的“缺失环节”。这种方法确保新手开发者能够通过主动构建而非被动观察，掌握工具包复杂的组装过程。MinGLE 以开源模板的形式按照 MIT 许可证发布，旨在成为 Geant4 生态系统中教育用途和新科学研究项目确定且可扩展的基础。

## 7. 致谢

本研究得到美国国家科学基金会（NSF）奖项 OIA-2437416、PHY-2411825 以及南达科他大学研究办公室的支持。支持本项目的计算工作在南达科他大学的高性能计算系统上完成，该系统由美国国家科学基金会奖项 OAC-1626516 资助。

## 8. 声明

在准备本研究期间，作者使用 Gemini 生成本文稿的初稿、润色语言并改进格式。使用该工具后，作者根据需要审阅并编辑了内容，并对已发表文章的内容承担全部责任。

## 参考文献

[1] S. Agostinelli 等，《Geant4——一种模拟工具包》，《核仪器与方法A辑》506 (3) (2003) 250–303。doi:10.1016/S0168-9002(03)01368-8。[2] J. Allison 等，《Geant4 的发展与应用》，《电气与电子工程师协会核科学汇刊》53 (1) (2006) 270–278，会议名称：《电气与电子工程师协会核科学汇刊》。doi:10.1109/TNS.2006.869826。[3] J. Allison 等，《Geant4 的最新进展》，《核仪器与方法A辑》835 (2016) 186–225。doi:10.1016/j.nima.2016.06.125。[4] J. Liu，《GEARS——一种可在运行时完全配置的 Geant4 应用程序》，arXiv.org[hep-ex, physics.comp-ph] (2025)。arXiv:2512.09246。[5] J. Liu，jintonic/mingle：Geant4迷你学习示例，https://github.com/jintonic/mingle（访问日期：2026年1月14日）。[6] GitHub, Inc.，GitHub，https://github.com（访问日期：2026年1月14日）。[7] L. Torvalds 等，Git 版本控制系统，https://git-scm.com（访问日期：2026年1月14日）。[8] S. Chacon、B. Straub，《Pro Git》第2版，Apress，2014年，第3章：Git 分支。网址：https://git-scm.com/book/en/v2[9] S. Chacon、B. Straub，《Pro Git》第2版，Apress，2014年，第2.6节：Git 基础——标签管理。网址：https://git-scm.com/book/en/v2

[10] J. Gruber，Markdown，https://daringfireball.net/projects/markdown（2004）。

[11] Kitware, Inc.，CMake 文档（访问日期：2026年1月14日）。网址https://cmake.org/documentation/

[12] Git for Windows 开发团队，Git for Windows（Git Bash）（访问日期：2026年1月14日）。网址https://gitforwindows.org/

[13] 微软公司，VisualStudio（访问日期：2026年1月14日）。网址https://visualstudio.microsoft.com/

[14] Physino -YouTube，https://www.youtube.com/@Physino（访问日期：2026年1月14日）。

[15] LLVM 项目，《JSON 编译数据库格式规范》（访问日期：2026年1月14日）。网址：https://clang.llvm.org/docs/JSONCompilationDatabase.html[16] 微软，Visual Studio Code（访问日期：2026年1月14日）。网址：https://code.visualstudio.com/[17] Neovim 团队，Neovim：高度可扩展的、基于 Vim 的文本编辑器（访问日期：2026年1月14日）。网址：https://neovim.io/[18] JetBrains s.r.o.，CLion：适用于 C 和 C++ 的跨平台集成开发环境 ++ （访问日期：2026年1月14日）。网址：https://www.jetbrains.com/clion/[19] Docker, Inc.，Docker 文档，https://docs.docker.com/（访问日期：2026年1月14日）。[20] J. Liu，physino/geant4 - Docker 镜像，https: // hub.docker.com/r/physino/geant4（访问日期：2026年1月14日）。[21] Geant4 协作组，Geant4 类 G4UIExecutive（访问日期：2026年1月14日）。网址：https://gitlab.cern.ch/geant4/geant4/ -/blob/master/source/interfaces/basic/include/G4UIExecutive.hh[22] Geant4 协作组，Geant4 类 G4UImanager（访问日期：2026年1月14日）。网址：https://gitlab.cern.ch/geant4/geant4/-/blob/master/source/intercoms/include/G4UImanager.hh

[23] Geant4 协作组，Geant4 类 G4RunManagerFactory（访问日期：2026年1月14日）。网址：https://gitlab.cern.ch/geant4/geant4/-/blob/master/source/run/include/G4RunManagerFactory.hh

[24] Geant4 协作组，Geant4 类 G4PhysListFactory（访问日期：2026年1月14日）。网址：https://gitlab.cern.ch/geant4/geant4/ -/blob/master/source/physics_lists/lists/include/G4PhysListFactory.hh[25] 来自文本文件的 Geant4 几何体，版本 1.0，https://geant4.web.cern.ch/ collaboration/workingi_groups/ persistency/docs/textgeom.pdf（访问日期：2026年1月14日）。[26] Geant4 协作组，Geant4 类 G4tgbVolumeMgr（访问日期：2026年1月14日）。网址：https://gitlab.cern.ch/geant4/geant4/ -/blob/master/source/persistency/ascii/include/G4tgbVolumeMgr.hh[27] Geant4 协作组，Geant4 基础示例 B1（访问日期：2026年1月14日）。网址：https://gitlab.cern.ch/geant4/geant4/-/tree/master/examples/basic/B1[28] R. Chytracek、J. Mccormick、W. Pokorski、G. Santin，《用于物理仿真和分析应用的几何描述标记语言》，电气与电子工程师协会《核科学汇刊》53 (5) (2006) 2892–2896。doi:10.1109/TNS.2006.881062。[29] Apache 软件基金会，Apache xerces-c++（访问日期：2026年1月14日）。网址：https://xerces.apache.org/xerces-c/[30] Geant4 协作组，Geant4 类 G4VisExecutive（访问日期：2026年1月14日）。网址：https://gitlab.cern.ch/geant4/geant4/ -/blob/master/source/visualization/management/include/G4VisExecutive.hh[31] 栈与堆 - LearnCpp.com （访问日期：2026 年 1 月 14 日）。网址：https://www.learncpp.com/cpp-tutorial/the-stack-and-the-heap/[32] Geant4 协作组，Geant4 类 G4GeneralParticleSource（访问日期：2026年1月14日）。网址：https://gitlab.cern.ch/geant4/geant4/-/blob/master/source/event/include/G4GeneralParticleSource.hh

[33] S. G. B. O'Connell、M. J. H. T. A. T. O'Connell, 通用粒子源, 《核仪器与方法A辑》535 (2004) 596–610。doi:10.1016/j.nima.2004.07.039。

[34] Geant4合作组, Geant4 类 G4ScoringManager （访问日期：2026年1月14日）。网址
https://gitlab.cern.ch/geant4/geant4/-/blob/master/source/digits_hits/detector/include/G4ScoringManager.hh[35] Geant4合作组, Geant4 类 G4TScoreNtupleWriter （访问日期：2026 年 1 月 14 日）。网址
https://gitlab.cern.ch/geant4/geant4/-/blob/master/source/digits_hits/utils/include/G4TScoreNtupleWriter.hh

[36] 维基百科, 元组 （访问日期：2026年1月14日）。网址https://en.wikipedia.org/wiki/Tuple

[37] ROOT团队, ROOT：TTree 类参考（访问日期：2026年1月14日）。网址
https://root.cern.ch/doc/master/classTTree.html

[38] ROOT 数据分析框架, https://root.cern.ch/ （访问日期：2026年1月14日）。

[39] J. Pivarski 等, Uproot：使用 Python 和 NumPy 读写 root 文件(访问日期：2026年1月14日)。网址
https://github.com/scikit-hep/uproot

[40] J. D. Hunter, Matplotlib：一种二维图形环境, 《科学与工程计算》9 (3) (2007) 90–95。doi:10.1109/MCSE.2007.55。

[41] Geant4 协作组, Geant4 类 G4AnalysisManager (访问日期：2026年1月14日)。网址
https://gitlab.cern.ch/geant4/geant4/-/blob/master/source/analysis/management/include/G4AnalysisManager.hh

[42] Geant4合作组, Geant4 类 G4TScoreHistFiller （访问日期：2026年1月14日）。
U
R
L
https://gitlab.cern.ch/geant4/geant4/-/blob/master/source/digits_hits/detector/include/G4TScoreHistFiller.hh[43] Geant4合作组, Geant4 类 G4GlobalMagFieldMessenger（访问日期：2026年1月14日）。URL
https://gitlab.cern.ch/geant4/geant4/-/blob/master/source/geometry/navigation/include/G4GlobalMagFieldMessenger.hh

[44] Geant4 用户指南：应用程序开发者指南：入门,
https://geant4-userdoc.web.cern.ch/UsersGuidesForApplicationDeveloper/html/GettingStarted/gettingStarted.html(访问日期：2026年1月14日)。