

Цикл обработки событий: Шаги

ЛЕКЦИЯ 9



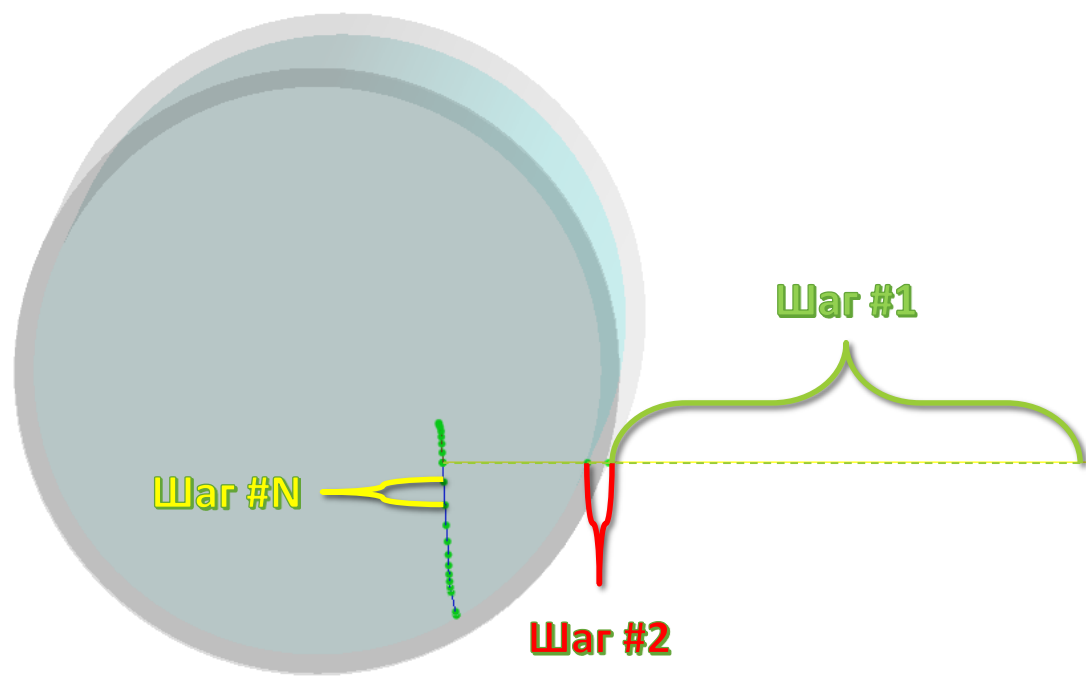


Содержание

- ❑ Шаги и **G4UserSteppingAction**
- ❑ Пример построения цикла обработки событий

Шаг

- ❑ Запуски состоят из событий
- ❑ События состоят из треков
- ❑ Треки состоят из шагов



Шаг — это минимальный элемент цикла обработки событий.

G4UserSteppingAction

Является опциональным базовым классом, для класса пользовательских «действий» в **КОНЦЕ** каждого шага. Данный класс не осуществляет моделирование шагов.

В данном классе предоставлен виртуальный метод :

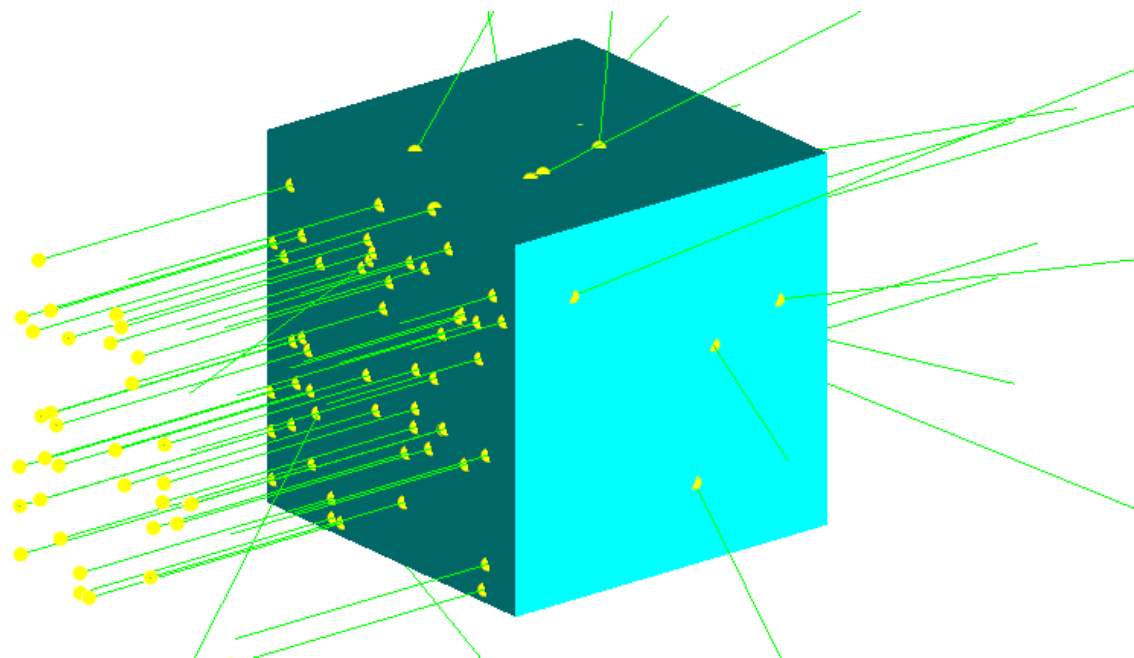
```
virtual void UserSteppingAction(const G4Step*)
```

Который и вызывается в конце каждого шага.

- ❑ Чтобы осуществить связь данного класса с классом обработки событий следует передать ему указатель на экземпляр класса обработки событий, к примеру:

```
14  class SteppingAction : public G4UserSteppingAction
15  {
16  public:
17      SteppingAction(EventAct*);
18
19      void UserSteppingAction(const G4Step*);
20  private:
21      EventAct*          EvAct;
22  };
```

Построение цикла обработки событий: Условие



Дано:

- ☐ Куб из материала **BGO**.
- ☐ Перпендикулярно к любой из граней падает гамма излучение **4.439** МэВ
- ☐ Определить энергосодержание в кубе



Построение цикла обработки событий:

Шаг 1.1

□ Унаследуем все классы обработки событий и подключим их.

□ Создадим шаблон класса действий для запусков унаследовав его от **G4UserRunAction**:

```
10 class RunAct: public G4UserRunAction{  
11     public:  
12         RunAct();  
13         ~RunAct();  
14         void BeginOfRunAction(const G4Run* aRun);  
15         void EndOfRunAction(const G4Run* aRun);  
16     };
```



Построение цикла обработки событий:

Шаг 1.2

- Создадим шаблон класса действий для событий унаследовав его от **G4UserEventAction**. Установим обратную связь данного класса с экземпляром класса действий для запусков, созданного на предшествующем этапе. Для этого передадим указатель на класс действий для запусков в конструкторе и будем хранить его в **private** поле:

```
RunAct* run;
```

Таким образом шаблон класса действий для событий:

```
11  ↩ class RunAct;
12
13  Ⓢ class EventAct: public G4UserEventAction{
14      public:
15          EventAct(RunAct* runAct);
16          ~EventAct();
17          void BeginOfEventAction(const G4Event* anEvent);
18          void EndOfEventAction(const G4Event* anEvent);
19
20      private:
21          RunAct* run;
22  };
```



Построение цикла обработки событий:

Шаг 1.3

- ❑ Создадим шаблон класса действий для шагов унаследовав его от **G4UserSteppingAction**. Установим обратную связь данного класса с экземпляром класса действий для событий, созданного на предшествующем этапе. Для этого передадим указатель на класс действий для событий в конструкторе и будем хранить его в **private** поле:

EventAct* event;

Таким образом шаблон класса действий для шагов:

```
11  class EventAct;
12
13  class StepAct: public G4UserSteppingAction{
14  public:
15      StepAct(EventAct* eventAct);
16      void UserSteppingAction(const G4Step*);
17
18  private:
19      EventAct* event;
20  };
```




Построение цикла обработки событий:

Шаг 1.4

□ Инициализируем все экземпляры созданных классов в методе **G4VUserActionInitialization::Build()**, передав куда нужно соответствующие указатели:

```
19 void Action::Build()const {  
20     SetUserAction(new PrimaryPat);  
21  
22     RunAct* runAct = new RunAct;  
23     SetUserAction(runAct);  
24  
25     EventAct* eventAct = new EventAct(runAct);  
26     SetUserAction(eventAct);  
27  
28     SetUserAction(new StepAct(eventAct));  
29  
30 }
```



Построение цикла обработки событий:

Шаг 2.1

Определим энерговыделение в кубе:

- Будем рассчитывать энерговыделение за один запуск, поэтому создадим в **RunAct** карту **std::map<G4double, G4int>** для хранения результатов

```
8 RunAct::RunAct() {  
9     result = new std::map<G4double, G4int>;  
10 }
```

и не забудем удалить её в деструкторе по завершению работы

```
16 RunAct::~RunAct() {  
17     delete result;  
18 }
```



Построение цикла обработки событий:

Шаг 2.2

Карта рассчитана на один запуск, поэтому будем обнулять её и заново заполнять каждый раз при вызове метода **RunAct::BeginOfRunAction()**:

```
16 void RunAct::BeginOfRunAction(const G4Run* aRun){
17     result->clear();
18     int nStep = 500;
19     G4double eMax = 5 * MeV;
20     for (int i = 0; i < nStep; i++)
21         result->insert(std::pair<G4double, G4int>(i * eMax / nStep, 0));
22 }
```

Примечание: в данном случае карта разделена на 500 ячеек до максимальной энергии в 5 МэВ

□ И предоставим метод по добавлению значений в данную карту (из событий)

```
29 void RunAct::AddEvent(G4double energy){
30     auto it = result->lower_bound(energy);
31     it->second++;
32 }
```

Примечание: std::map::lower_bound возвращает указатель на первый элемент карты, чей ключ меньше передаваемого



Построение цикла обработки событий:

Шаг 3

□ Осуществим регистрацию энергосвечения в **StepAct**

Метод **StepAct::UserSteppingAction()** вызывается в конце каждого шага. Чтобы отсечь события энергосвечения которые прошли за пределами исследуемого объема определим материал места где произошла энергопотеря.

`aStep->GetTrack()->GetVolume()->GetLogicalVolume()->GetMaterial()->GetName()`:

Примечание: из текущего шага определяем какому треку он принадлежит, для трека определяем в каком физическом объеме мы оказались, из физического получаем логический объем, для логического объема – материал, а для материала его имя.

Для шага будем определять энергопотерю на нем и передавать её в наше событие.

```
14 void StepAct::UserSteppingAction(const G4Step * aStep) {  
15     if(aStep->GetTrack()->GetVolume()->GetLogicalVolume()->GetMaterial()->GetName()=="G4_BG0")  
16         event->AddEnDep(aStep->GetTotalEnergyDeposit());  
17 }
```



Построение цикла обработки событий:

Шаг 4

□ Создадим для класса **EventAct** переменную в которой мы будем суммировать энерговыделение со всех шагов в рамках одного события. В начале каждого события (т.е. в методе **EventAct::BeginOfEventAction()**) мы будем обнулять значение этой переменной:

```
14 void EventAct::BeginOfEventAction(const G4Event *anEvent) {  
15     EnergyDep = 0;  
16 }
```

А в конце каждого события **EventAct::EndOfEventAction()** мы будем передавать это значение в карту в запуске для заполнения^

```
10 void EventAct::EndOfEventAction(const G4Event *anEvent) {  
11     run->AddEvent(EnergyDep);  
12 }
```

Изменять же это значение мы будем **ВО ВРЕМЯ** события из «нужных» шагов за счет дополнительного метода:

```
26 void EventAct::AddEnDep(G4double en){  
27     EnergyDep+=en;  
28 }
```



Построение цикла обработки событий: Итоги

- ❑ В **StepAct** мы определяем было ли энергосвечение на данном шаге и произошло ли данное событие в исследуемом объеме. Полученное значение передаем в **EventAct** через метод **EventAct::AddEnDep()**
- ❑ В **EventAct** мы создаем переменную для накопления полного энергосвечения за одно событие. В начале события мы обнуляем данную переменную. В конце события мы передаем её в **RunAct**.
- ❑ В **RunAct** мы создаем карту в которой мы в нужный канал по полученному полному энергосвечению с каждого события добавляем единицу, получая таким образом модельный наборный спектр.

Примечание: Как вариант данный спектр можно сохранить в отдельный файл в финале запуска

```
25 void RunAct::EndOfRunAction(const G4Run* aRun){  
26     std::fstream fout("../res.txt",std::ios::out);  
27     for (auto it: *result)  
28         fout<<it.first<<" | " <<it.second <<'\n';  
29     fout.close();  
30 }
```

Приложение: StepAct.hh и StepAct.cc

StepAct.hh

```
11  ↔ class EventAct;
12
13  ⚙ class StepAct: public G4UserSteppingAction{
14      public:
15      ↔   StepAct(EventAct* eventAct);
16      ✖   void UserSteppingAction(const G4Step*);
17
18      private:
19      EventAct* event;
20  };
```

StepAct.cc

```
10  ↔ StepAct::StepAct(EventAct *eventAct):event(eventAct) {
11
12  }
13
14  ✖ void StepAct::UserSteppingAction(const G4Step * aStep) {
15      if(aStep->GetTrack()->GetVolume()->GetLogicalVolume()->GetMaterial()->GetName()=="G4_BGO")
16          event->AddEnDep(aStep->GetTotalEnergyDeposit());
17  }
```

Приложение: EventAct.hh

```
11  ↩ class RunAct;
12
13  ☒ class EventAct: public G4UserEventAction {
14  public:
15  ↩   EventAct(RunAct *runAct);
16
17  ☒   void BeginOfEventAction(const G4Event *anEvent);
18
19  ☒   void EndOfEventAction(const G4Event *anEvent);
20
21  ↩   void AddEnDep(G4double en);
22
23  private:
24   RunAct *run;
25   G4double EnergyDep;
26  ☒ };
```


Приложение: EventAct.cc

```
10  ↩ EventAct::EventAct(RunAct *runAct):run(runAct) {
11
12  }
13
14  ✕ ↩ void EventAct::BeginOfEventAction(const G4Event *anEvent) {
15      EnergyDep = 0;
16  }
17
18  ✕ ↩ void EventAct::EndOfEventAction(const G4Event *anEvent) {
19      run->AddEvent(EnergyDep);
20  }
21
22  ↩ void EventAct::AddEnDep(G4double en) {
23      EnergyDep += en;
24  }
```

Приложение: RunAct.hh

```
10 class RunAct: public G4UserRunAction {  
11     public:  
12         RunAct();  
13  
14         ~RunAct();  
15  
16         void BeginOfRunAction(const G4Run *aRun);  
17  
18         void EndOfRunAction(const G4Run *aRun);  
19  
20         void AddEvent(G4double energy);  
21  
22     private:  
23         std::map<G4double, G4int> *result;  
24     };
```

Приложение: RunAct.cc часть 1

```
9  RunAct::RunAct() {  
10      result = new std::map<G4double, G4int>;  
11  }  
12  
13  RunAct::~RunAct() {  
14      delete result;  
15  }  
16  
17  void RunAct::BeginOfRunAction(const G4Run* aRun) {  
18      result->clear();  
19      int nStep = 500;  
20      G4double eMax = 5 * MeV;  
21      for (int i = 0; i < nStep; i++)  
22          result->insert(std::pair<G4double, G4int>(i * eMax / nStep, 0));  
23  }  
24
```

Приложение: RunAct.cc часть 2

```
25 void RunAct::EndOfRunAction(const G4Run* aRun) {  
26     std::fstream fout("../res.txt", std::ios::out);  
27     for (auto it: *result)  
28         fout << it.first << " | " << it.second << '\n';  
29     fout.close();  
30 }  
31  
32 void RunAct::AddEvent(G4double energy) {  
33     auto it = result->lower_bound(energy);  
34     it->second++;  
35 }
```

Приложение: Action::Build()

```
19 void Action::Build()const {  
20     SetUserAction(new PrimaryPat);  
21  
22     RunAct* runAct = new RunAct;  
23     SetUserAction(runAct);  
24  
25     EventAct* eventAct = new EventAct(runAct);  
26     SetUserAction(eventAct);  
27  
28     SetUserAction(new StepAct(eventAct));  
29  
30 }
```