Открытый программный код МС транспорта фотонного и электронного излучения методом Монте-Карло

Горлачев Г.Е., РОНЦ

Далечина А.В., Центр Гамма-Нож

Meetup 23 мая 2017 г.

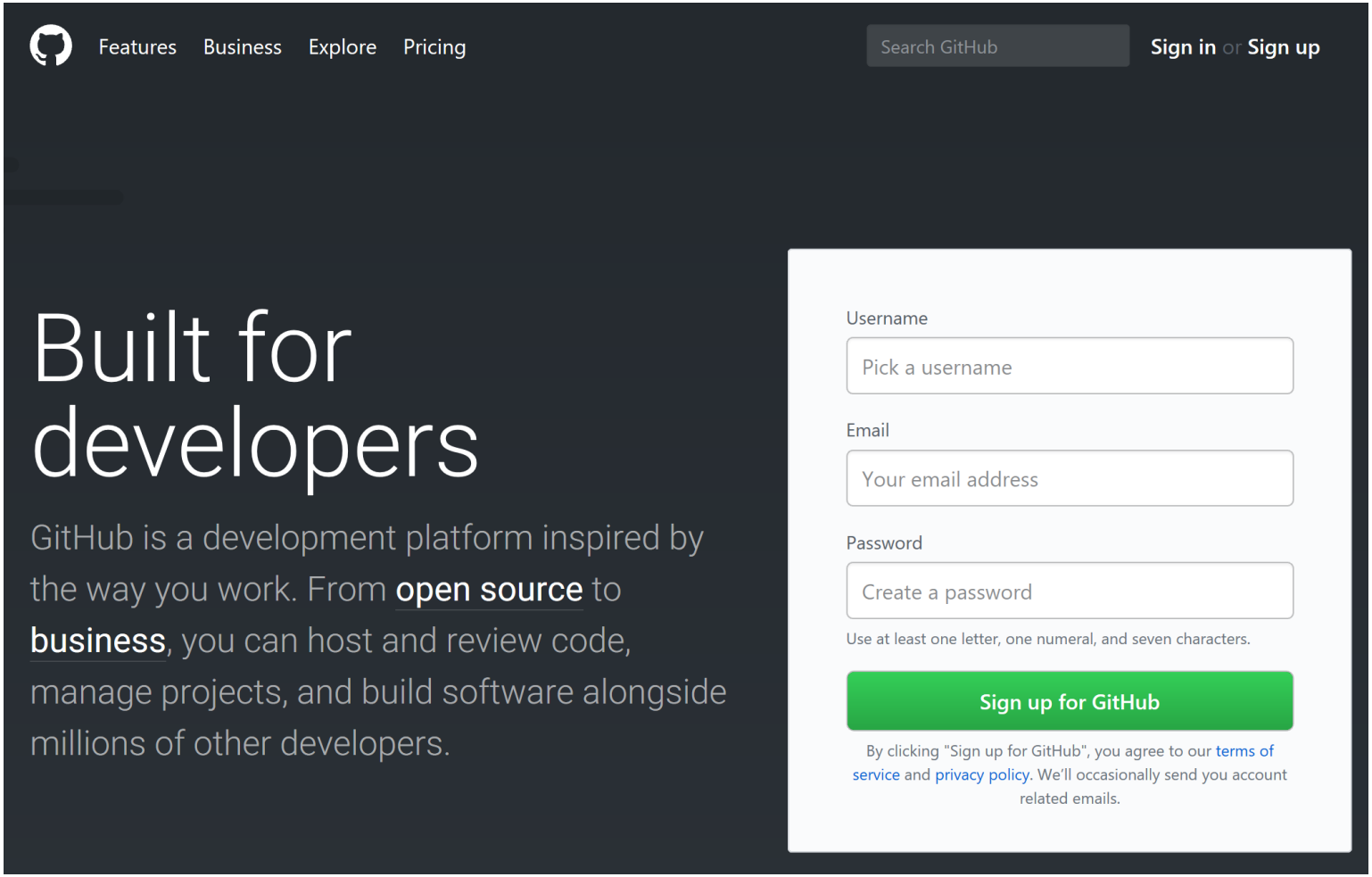
Введение

- Метод Монте-Карло в лучевой терапии – основной метод дозиметрических расчетов
- Код MC родился в процессе работы над задачами в предметной области на протяжении 20 лет
- Может быть интересен и в учебных целях благодаря реализации в строгой понятной философии объектно-ориентированного программирования на языке C++

GitHub

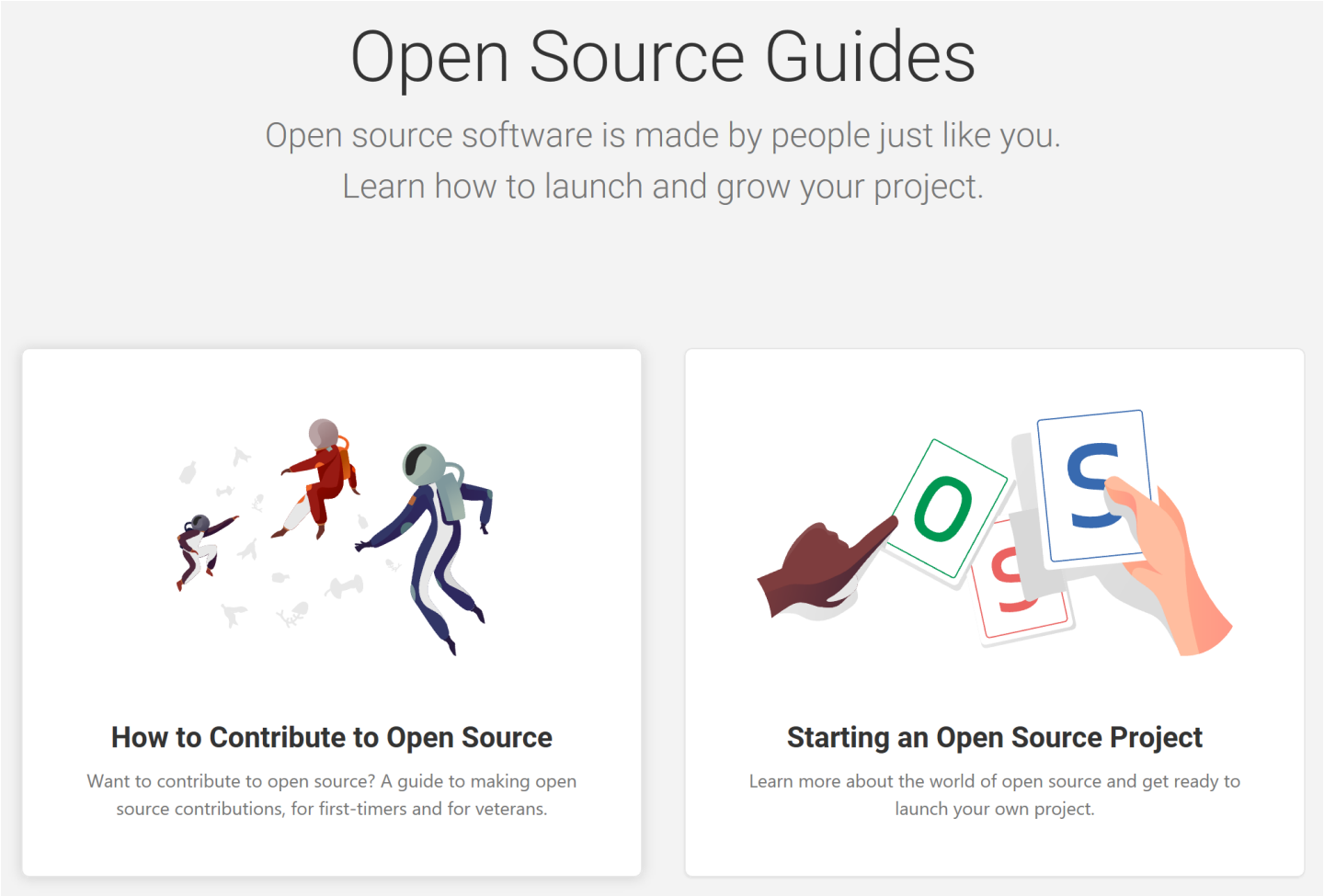
Платформа для разработок

https://github.com/



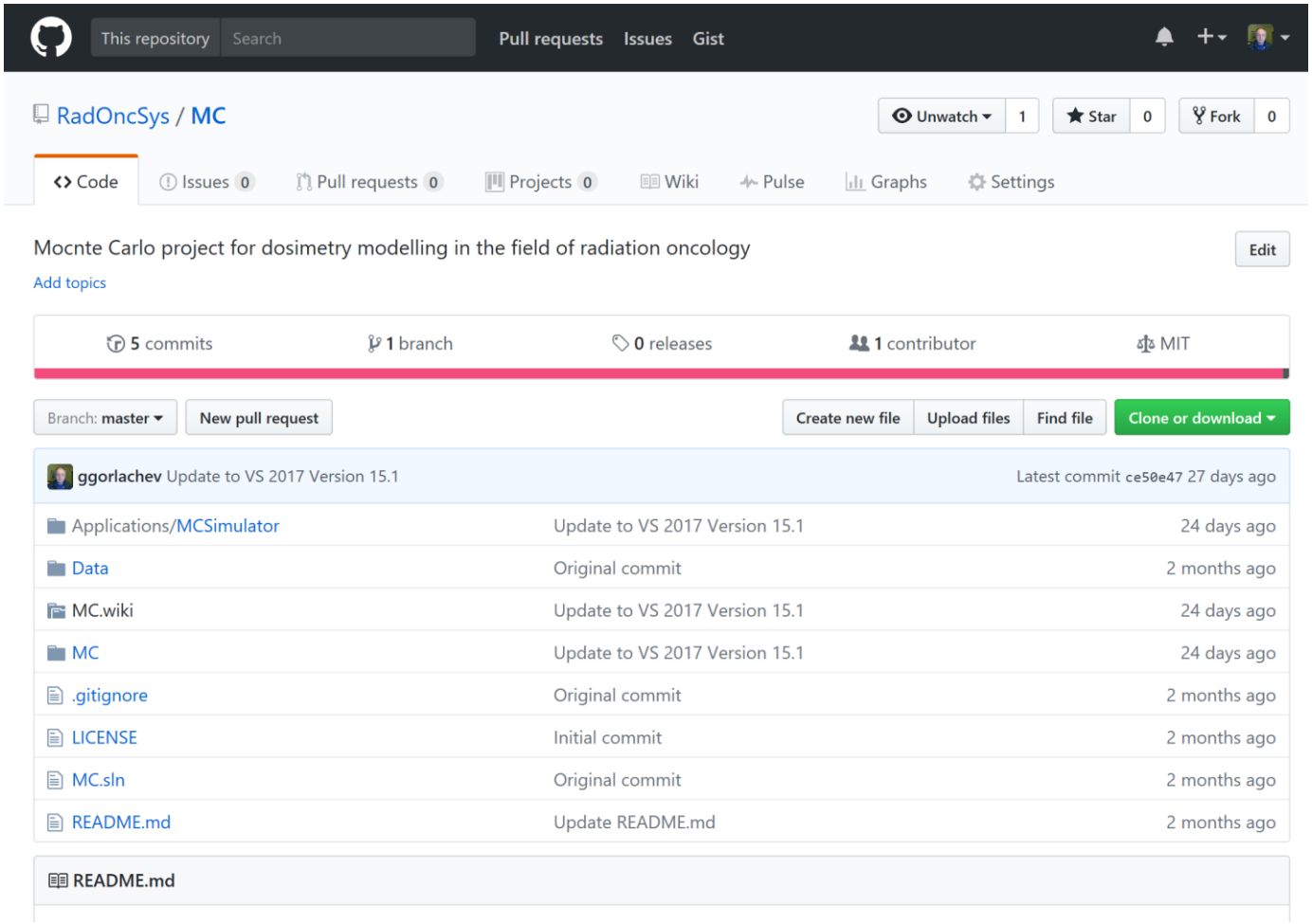
GitHub – это платформа для разработок. С ее помощью вы можете осуществлять как коммерческие, так и открытые разработки, управлять проектами параллельно с миллионами других разработчиков.

https://opensource.guide/



Полезный ресурс для понимания философии открытых кодов

https://github.com/RadOncSys/MC



Домашняя страница кода MC

EGS-Nova

Предшественник кода MC

http://rcwww.kek.jp/research/egs/epub/aap/js3nov98.html

EGS-Nova: An Adaptation of EGS in C/C++

Dear EGS4 Users,

Version 0.1.0 of Nova is now available at

ftp://stereo.medphysics.nemc.org/pub/Nova

There is also now a Nova website at URL

http://www.nemc.org/nova/

James C. Satterthwaite, Ph.D.
Department of Radiation Oncology
New England Medical Center #246
750 Washington Street
Boston, MA 02111
Voice: 617-636-0612
Fax: 617-636-7621
james.satterthwaite@es.nemc.org

HenHouse Roosters
Last revised 3-Nov-1998

Ссылка на сайт кода Nova.
Настоящий статус проекта не известен.

Другие коды транспорта методом Монте-Карло

EGSNrc, Penelope, MCNP, Geant, …

МК код	Доступность
MCNP	https://mcnp.lanl.gov/ **request Radiation Safety Information Computational Center (RSICC)**
EGS4	http://rcwww.kek.jp/research/egs/
EGS nrc	http://nrc-cnrc.github.io/EGSnrc/
EGS 5	http://rcwww.kek.jp/research/egs/egs5.html
Penelope	https://www.oecd-nea.org/tools/abstract/detail/nea-1525 **request programs@oecd-nea.org**
Geant 4	https://geant4.web.cern.ch/geant4/support/download.shtml
Fluka	http://www.fluka.org/fluka.php **register user**

Структура кода МС

Библиотеки, приложения, физика, геометрия, статистика, визуализация, многопоточность.

Структура проектов в Visual Studio

Demo

- Навигация по проектам
- Состав библиотеки транспорта

Транспорт частиц

Разыгрывание взаимодействия фотонов, приближение непрерывных потерь заряженных частиц, пересечение границы

Класс описания частицы (mcParticle)

```
class mcParticle
{
public:
        mcParticle(void);
        mcParticle(mc_particle_t pt, int pq, double pke, const geomVector3D& pp, const geomVector3D& pu);
        mcParticle(const mcParticle& p);
        ~mcParticle(void);

public:
        enum mc_particle_t t;
        int q;
        double ke;
        geomVector3D p;
        geomVector3D plast;// Точка последнего взаимодействия
        geomVector3D u;
        double dnear;
        mcRegionReference region;
        double weight;

        // Расстояние до следующего взаимодействия в единицах длин среднего пробега
        double mfps;

        // Относительная плотность региона, в котором находится частица в данный момент
        double regDensityRatio;

        // Указатель на транспортный объект,
        // в котором частица находится в данный момент
        mcTransport* transport_;

        // Место рождения частицы
        //mcRegionReference reg_born;

        // Место последнего взаимодействия
        //mcRegionReference reg_last_scatter;

        // Указатель на объект, собирающий перемещения частиц между модулями
        mcScoreTrack* trackScore_;
```

Помимо физических свойств частицы класс содержит ряд вспомогательных переменных, позволяющий проводить гибкий анализ.

Класс транспорта частицы (mcTransport)

```
class mcTransport : public mcObj
{
        void setPosition(const geomVector3D& orgn, const geomVector3D& z, const geomVector3D& x);
        void setMediaRef(const mcMedia* media);

        virtual void beginTransport(mcParticle& p);
        virtual void beginTransportInside(mcParticle& p);
        virtual void endTransport(mcParticle* particle);

        static void simulate(mcThread* thread);

        virtual mc_move_result_t moveParticle(mcParticle* particle, double& step, double& edep);

        void setScore(mcScore* score);

        void setPreviousTransport(mcTransport* t) { previousTransport_ = t; }
        void setNextTransport(mcTransport* t) { nextTransport_ = t; t->setPreviousTransport(this); }
        void setInternalTransport(mcTransport* t);
        void setExternalTransport(mcTransport* t);

        // Преобразования координат
        const geomMatrix3D& MW2T() { return mwtot_; }
        const geomMatrix3D& MT2W() { return mttow_; }

        // Разыгрывание пути
        static double HowManyMFPs(mcRng& rng);

        virtual void dump(ostream& os) const;
        virtual void dumpVRML(ostream& os) const;

        //
        // Геометрия (методы нужны для поддержки стандартной функции moveParticle)
        //
        virtual double getDistanceInside(mcParticle& p) const;
        virtual double getDistanceOutside(mcParticle& p) const;
        virtual double getDNearInside(const geomVector3D& p) const;
};
```

Ядро транспорта. Поддержка геометрических модулей реализована через наследование базового класса транспорта и перегрузку функций расстояния до границы.

mcTransport::simulate

```
void mcTransport::simulate(mcThread* thread)
{
  mcParticle** pCurParticle = thread->CurrentParticle();
  mcParticle* particleStack = thread->ParticleStack();

  while ((*pCurParticle) >= particleStack)
  {
    // Указатель на транспортный объект,
    // в котором частица находится в данный момент
    mcTransport* t = (*pCurParticle)->transport_;

    // Сохраняем стартовую точку для скоринга трэка.
    // Конечная точка хранится в текущей частице.
    geomVector3D point((*pCurParticle)->p);
    enum mc_particle_t pt = (*pCurParticle)->t;
    mcRegionReference region = (*pCurParticle)->region;
    double weight = (*pCurParticle)->weight;

    if ((*pCurParticle)->mfps <= 0)
      (*pCurParticle)->mfps = HowManyMFPs(thread->rng());

    double step, edep;//step & edep???
    mc_move_result_t mres = t->moveParticle(*pCurParticle, step, edep);

    if (mres == MCMR_DISCARGE)
    {
      if (edep > 0 && t->score_ != nullptr)
        t->score_->ScorePoint(edep * weight, thread->id(), region, pt, point);
      continue;
    }

    if (edep > 0 && t->score_ != nullptr)
      t->score_->ScoreLine(edep * weight, thread->id(),
                  region, pt, point, (*pCurParticle)->p);

    // Если частица после израсходования пути остается в транспорте,
    // то разыгрываем взаимодействие.
    // Если частица покидает транспорт, берем следующую частицу из стека
    // или прерываем симуляцию.

    if (mres == MCMR_INTERUCT)
    {
      const mcPhysics* phys = t->media_->getPhysics((*pCurParticle)->t);
      const mcMedium* med = t->media_->getMedium((*pCurParticle)->t,
                            (*pCurParticle)->region.medidx_);

      point = (*pCurParticle)->p;
      pt = (*pCurParticle)->t;
      region = (*pCurParticle)->region;
      weight = (*pCurParticle)->weight;

      // Частицы с энергией ниже критической должны быть уничтожены
      // раньше любых расчетов транспорта.
      if (phys->Discarge(*pCurParticle, *med, edep))
      {
        if (edep > 0 && t->score_ != nullptr)
          t->score_->ScorePoint(edep * weight, thread->id(), region, pt, point);
        continue;
      }

      edep = phys->DoInterruction((*pCurParticle), med);
      if (edep > 0 && t->score_ != nullptr)
        t->score_->ScorePoint(edep * weight, thread->id(), region, pt, point);
      if ((*pCurParticle)->ke == 0)
        thread->RemoveParticle();
      else
        (*pCurParticle)->mfps = 0;

        continue;
      }
      else if (mres == MCMR_EXIT)
      {
        t->endTransport(*pCurParticle);
        continue;
      }
      else if (mres == MCMR_CONTINUE)
      {
        // По некоторым причинам частица осталась в стеке нужно
        // продолжить ее транспорт.
        // Например, частица переместилась без изменения энергии
        // на поверхность объекта.
        continue;
      }
      else
        throw std::exception("Unexpected particle move result in simulator");
    }
}
```

mcTransport::moveParticle

```
mc_move_result_t mcTransport::moveParticle(mcParticle* particle, double& step, double& edep)
{
edep = 0;

// HACK!!
// По непонятным причинам координаты частицы могут быть абсурдными.
// Удалаяем такие частицы
if (_isnan(particle->p.x()) != 0)
{
//cout << "Non number position or direction in object: " << this->getName() << endl;
cout << "Non number position in object: " << this->getName() << endl;
cout << "Position: " << particle->p;
cout << "Direction: " << particle->u;
particle->thread_->RemoveParticle();
return MCMR_DISCARGE;
}

// Переместить частицу на поверхность, если она еще не внутри
step = DBL_MAX;
if (particle->region.idx_ == 0)
{
int iregion = 0;
if (isMultiRegions_)
{
for (unsigned i = 0; i < regions_.size(); i++)
{
double f = regions_[i]->getDistanceOutside(*particle) + DBL_EPSILON;
if (f < step)
{
iregion = i;
step = f;
}
}
}
// Если установлен флаг типа пересекаемой поверхности как внутренней,
// То определенно речь не о повторной возможности входа в данный объект,
// а о переходе в следующий
else if (particle->exitSurface_ != mcParticle::temb_shit_t::Internal)
step = getDistanceOutside(*particle) + DBL_EPSILON;

if (step == DBL_MAX) { // промазали, летим в следующий слой
endTransport(particle);
return MCMR_CONTINUE;
}

// При необходимости запоминаем отрезок в мировой системе координат
if (particle->trackScore_)
particle->trackScore_->score(particle->thread_->id(), particle->t, particle->p * mttow_, (particle->p + (particle->u * step)) * mttow_, particle->ke);

particle->p += particle->u * step;
particle->dnear = 0;
particle->region.idx_ = iregion + 1;

if (isMultiRegions_)
{
particle->regDensityRatio = regions_[iregion]->getDefDensity();
particle->region.medidx_ = regions_[iregion]->getDefMedIdx();
}
else
{
particle->regDensityRatio = defdensity_;
particle->region.medidx_ = defmedidx_;
}

// Возвращаемся, чтобы повторить шаг.
// В противном случае шаг до поверхности будет включен в
// потери энергии заряженной частицы.
if (step > DBL_EPSILON)
return MCMR_CONTINUE;
}

const mcPhysics* phys = media_->getPhysics(particle->t);
const mcMedium* med = media_->getMedium(particle->t, particle->region.medidx_);//Параметры сред транспорта фотонов и электронов

// Частицы с энергией ниже критической должны быть уничтожены раньше любых расчетов транспорта.
if (phys->Discarge(particle, *med, edep))
return MCMR_DISCARGE;

double freepath = phys->MeanFreePath(particle->ke, *med, defdensity_);
step = freepath * particle->mfps;

if (step < particle->dnear)
{
double stepRequested = step;
edep = phys->TakeOneStep(particle, *med, step);

// Сохраняем фрагмент трека после шага, который может меняться в процессе последнего.
// К тому же координаты точки имеют значение после шага. Поэтому в следующих расчетах шаг отрицательный.
if (particle->trackScore_)
particle->trackScore_->score(particle->thread_->id(), particle->t, particle->p * mttow_, (particle->p - (particle->u * step)) * mttow_, particle->ke);

  // Шаг может быть меньше запрошенного только в случае заряженных частиц
// и означает, что просто выполнен шаг при нерерывных потерях энергии и рассения
// и дискретного события не случилось.
if (step < stepRequested)
return MCMR_CONTINUE;
else
return MCMR_INTERUCT;
}


  double dist;
if (isMultiRegions_)
{
dist = regions_[particle->region.idx_ - 1]->getDistanceInside(*particle) + DBL_EPSILON;
}
else
{
dist = getDistanceInside(*particle) + DBL_EPSILON;// Новый трик с тем, чтобы частица чуть-чуть заступала за границу;
}

// HACK!!
// По непонятным причинам частицы могут быть за пределами объекта,
// хотя транспорт как внутри. До выяснения причин вывод сообщений о подобных событиях.
if (dist == DBL_MAX || dist < -0.01)
{
cout << "Wrong particle transport inside object:" << this->getName() << endl;
cout << "Position: " << particle->p;
cout << "Direction: " << particle->u;
cout << "Dnear: " << particle->dnear << endl;
cout << "Distance = " << dist << endl;
particle->thread_->RemoveParticle();
return MCMR_DISCARGE;
}

if (step < dist)
{
double stepRequested = step;
edep = phys->TakeOneStep(particle, *med, step);

if (particle->trackScore_)
particle->trackScore_->score(particle->thread_->id(), particle->t, particle->p * mttow_, (particle->p - (particle->u * step)) * mttow_, particle->ke);

if (isMultiRegions_)
particle->dnear = regions_[particle->region.idx_ - 1]->getDNearInside(particle->p);
else
particle->dnear = getDNearInside(particle->p);
if (step < stepRequested)
return MCMR_CONTINUE;
else
return MCMR_INTERUCT;
}
else
{
// HACK! На поверхности возможно залипание, если расстояние в пределах погрешности вычислений.
static const double epsln = DBL_EPSILON * 10;
if (dist < epsln)
dist = epsln;
step = dist;
edep = phys->TakeOneStep(particle, *med, step);

if (particle->trackScore_)
particle->trackScore_->score(particle->thread_->id(), particle->t, particle->p * mttow_, (particle->p - (particle->u * step)) * mttow_, particle->ke);

particle->mfps -= step / freepath;
if (step == dist)
{
// !! Для поддержки композитных модулей и модулей с невыпуклыми поверхностями
// (например, цилиндрические кольца) мы не устанавливаем шаг выхода а меняем индес региона на 0.
// В этом случае данная функция будет вызвана вновь и предпринята попытка продолжить транспорт в модуле.
// В случае отсутствия попадания частица будет, наконец, передана следующему транспорту.
particle->region.idx_ = 0;
}
return MCMR_CONTINUE;
}
}
```

Старт симуляции

```
for (int ib = 0; ib < nBanches; ib++)
{
	wcout << L"Time:" << TIMESINCE(simStartTime) << L" sec«
		<< L"	Start bunch " << ib + 1 << L" of " << nBanches << endl;

	concurrency::parallel_for(0u, (unsigned)nThreads,
		[threads, source, tstart, nParticles, energySource, startinside](unsigned it)
	{
		mcParticle particle;

		for (int ii = 0; ii < nParticles; ii++)
		{
			source->sample(particle, &threads[it]);
			energySource[it] += particle.ke;

			if (startinside)
				tstart->beginTransportInside(particle);
			else
				tstart->beginTransport(particle);
		}
	});
}
```

Симуляция начинается в основной программе самплингом частицы из источника.
В зависимости от настраиваемых условий частица передается на симуляцию в первый объект внутри него или снаружи.

Многопоточность (Multithreading)

```
class mcThread
{
public:
      mcThread();
      ~mcThread(void);

      int id() const { return threadIdx_; }
      void setId(int id);

      mcRng& rng() { return rng_; }

      mcParticle* NextParticle();
      void RemoveParticle();
      mcParticle* DuplicateParticle();

      mcParticle** CurrentParticle() { return &pCurParticle_; }
      mcParticle* ParticleStack() { return particleStack_; }

protected:
      int threadIdx_;
      mcRng rng_;
      mcParticle* pCurParticle_;
      mcParticle* particleStack_;
};
```

Многопоточность обеспечивается поддержкой копий транспорта и скоринга пропорционально количеству ядер компьютера. Инфраструктура скрывает параллелизм от пользователя. Классы скоринга при выводе результатов автоматически сливают данные всех потоков.

Линейная логика связи геометрий

Упрощение описания задачи за счет ограничения объектов, куда частица может попадать вылетая из очередного объекта

Схема системы формирования пучка излучения

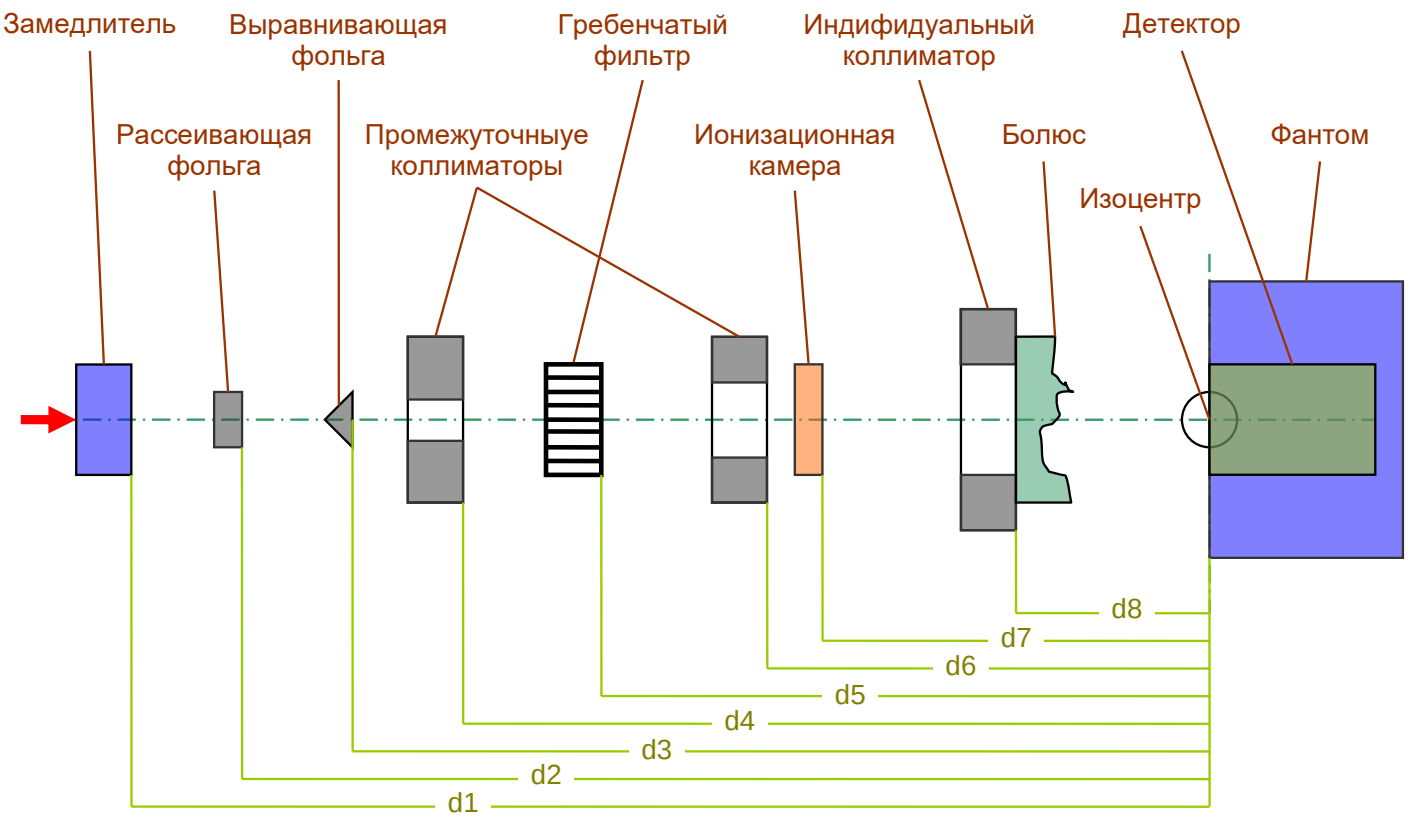
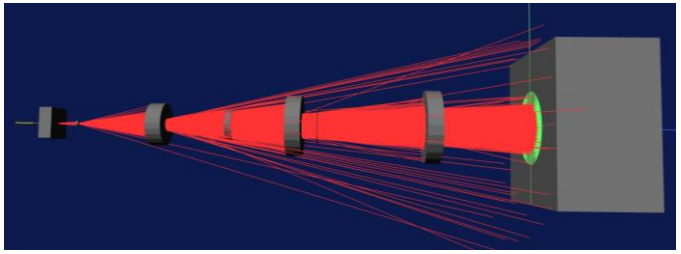


Рис. 1 *Схема пассивной системы формирования терапевтических полей облучения, использующей двойное рассеивание и гребенчатые фильтры.*

Как правило объекты транспорта выстроены в линейную цепочку, что упрощает описание геометрии



Организация переходов по цепочке (mcTransport::beginTransport)

```
void mcTransport::beginTransport(mcParticle& p)
{
	if (this->media_ == nullptr)
		throw std::exception((string("Media not set in the module \"") + this->getName() + "\"").c_str());

		// Указатель на транспортный объект, в котором частица находится в данный момент
		p.transport_ = this;

		// Объекты транспорта могут ставить печать в момент
		// BeginTransport обозначая, что частица в них находилась.
		p.regionFlags |= stamp_;

		mcParticle* particle = p.thread_->NextParticle();
		*particle = p;
		particle->p = p.p * mwtot_;
		particle->plast = p.plast * mwtot_;
		particle->u = particle->u.transformDirection(mwtot_);
		particle->dnear = 0;
		particle->mfps = HowManyMFPs(p.thread_->rng());

		particle->region.idx_ = 0;
		particle->region.medidx_ = 0;
		particle->regDensityRatio = DBL_EPSILON;

		if (score_)
			score_->ScoreFluence(*particle);

		// Транспорт в локальной системе координат
		simulate(p.thread_);
}
```

Симуляция осуществляется объектом класса транспорта. Частица регистрируется в стеке. Устанавливаются некоторые ее параметры связанные с объектом транспорта (указатели на сопутствующие данные, преобразование координат в систему объекта транспорта).

Организация переходов по цепочке (mcTransport::endTransport)

```
void mcTransport::endTransport(mcParticle* particle)
{
	if (particle->exitSurface_ == mcParticle::temb_shit_t::Undefined && (externalTransport_ != nullptr || internalTransport_ != nullptr))
		throw std::exception("All Get Distance functions of embedded transports should take care  ...");

	else if (particle->exitSurface_ == mcParticle::temb_shit_t::Internal && internalTransport_ == nullptr)
		throw std::exception("It should not be possible to hit internal surface if internal object does not exist or not indicated");

	mcParticle pp = **particle->thread_->CurrentParticle();
	particle->thread_->RemoveParticle();
	if (pp.ke == 0) return;

	pp.p = pp.p * mttow_;
	pp.plast = pp.plast * mttow_;
	pp.u = pp.u.transformDirection(mttow_);

	// Если пересекаем внешнюю поверхность и нет охватывающего объекта,
	// то мы просто передаем управления стандартной однонитиевой цепочке объектов.
	if ((particle->exitSurface_ == mcParticle::temb_shit_t::External && externalTransport_ == nullptr) ||
		(externalTransport_ == nullptr && internalTransport_ == nullptr))
	{
		if (pp.u.z() < 0 && previousTransport_ != nullptr)
			previousTransport_->beginTransport(pp);
		else if (pp.u.z() > 0 && nextTransport_ != nullptr)
			nextTransport_->beginTransport(pp);
		else if (pp.trackScore_)
			pp.trackScore_->score(particle->thread_->id(), pp.t, pp.p, pp.p + (pp.u * 100), pp.ke);
	}
	else
	{
		if (particle->exitSurface_ == mcParticle::temb_shit_t::Internal && internalTransport_ != nullptr)
			internalTransport_->beginTransportInside(pp);
		else if (particle->exitSurface_ == mcParticle::temb_shit_t::External && externalTransport_ != nullptr)
			externalTransport_->beginTransportInside(pp);
		else if (pp.trackScore_)
			pp.trackScore_->score(particle->thread_->id(), pp.t, pp.p, pp.p + (pp.u * 100), pp.ke);
	}
}
```

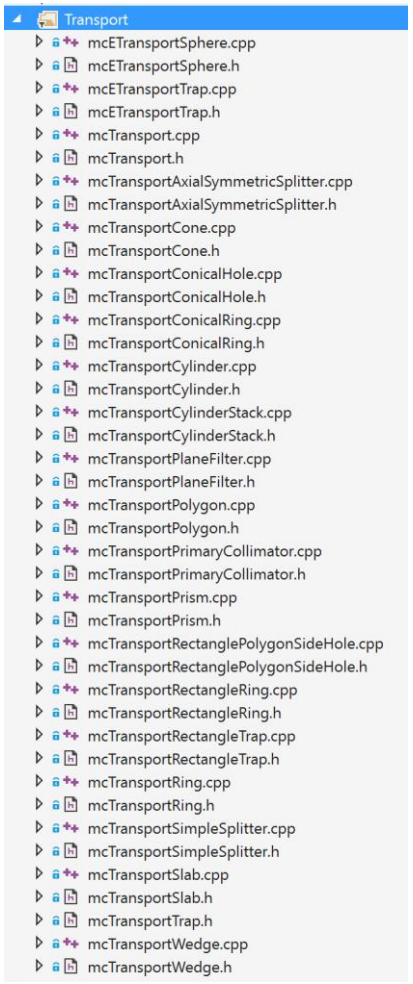
В момент вылетания частицы из объекта на основании направления движения выбирается объект, в который она может попасть. Если такового не находится, то она просто исчезает.

Геометрические модули

Пример программирования геометрии – профилированный коллиматор радиотерапевтической установки

Геометрический модуль – наследник класса транспорта

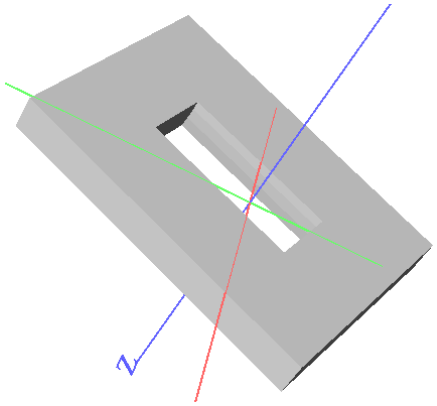
```
// Класс прямоугольного отверстия в параллелепипеде.
// Класс создан для описания основного коллиматора Рокуса-Р.
// В собственной системе координат центр последней находится в плоскости входного отверстия.
// Ось Z направлена от источника в сторону изоцента (для лучшего понимания).
// Положение шторок описывается как симметричное.
// Ассиметричные шторки могут симулироваться смещением центра системы координат объекта.
class mcTransportRectanglePolygonSideHole : public mcTransport
{
public:
      mcTransportRectanglePolygonSideHole(const geomVector3D& orgn,
            const geomVector3D& vz,
            const geomVector3D& vx,
            double dx, double dy, std::vector<double>& z,
            std::vector<double>& x, std::vector<double>& y);
      virtual ~mcTransportRectanglePolygonSideHole(void);

      void dump(ostream& os) const override;
      void dumpVRML(ostream& os)const override;

      double getDistanceInside(mcParticle& p) const override;
      double getDistanceOutside(mcParticle& p) const override;
      double getDNearInside(const geomVector3D& p) const override;

protected:
      void dumpVRMLPolygonSideHole(ostream& os) const;

      ...
};
```



Для описания нового геометрического модуля достаточно написать конструктор, виртуальные функции определения расстояний до границ и функции самопредставления (внутренние данные и VRML).

Коллиматор поля (mcTransportRectanglePolygonSideHole)

Профилированный прямоугольный коллиматор



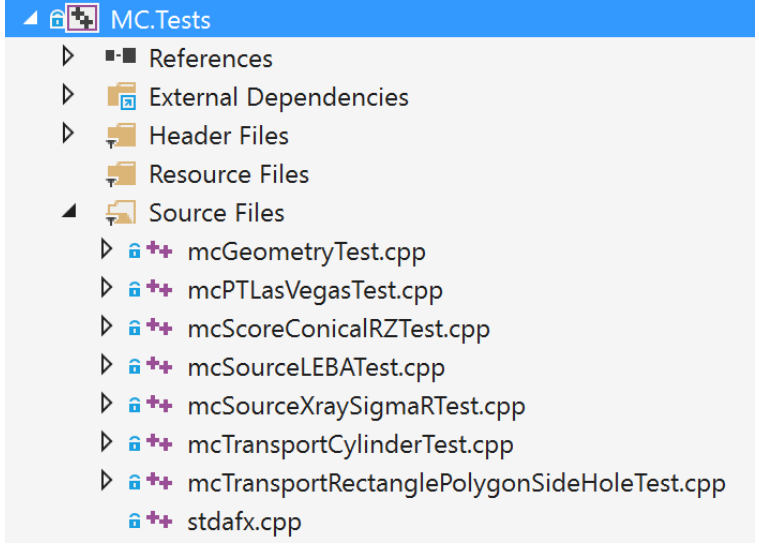
Представляет прямоугольное отверстие внутри параллелепипеда, внутренняя часть которого описывается полигоном. Позволяет задавать толщину объекта, ширину камней коллиматора, положение верхней границы камней коллиматора по осям X и Y. Объект полностью воспроизводит основной коллиматор аппарата Рокус-Р. Ассиметричные поля формируются смещением центра объекта.

Пример описания объекта при симуляции:

```
<module type="rectanglepolygonsidehole" name="Main collimator" medium="W700ICRU"
density="1">
  <Color r="0.8" g="0.8" b="0.8" t="0.0" />
  <position unit="cm" x="-5.0" y="3" z="0" />
  <normal x="0" y="0" z="1" />
  <xaxis x="1" y="0" z="0" />
  <size unit="cm" d="15.5" x0="6.0" y0="15.0"/>
  <polygon>
    <point x="0" z="0"/>
    <point x="0" z="2.0"/>
    <point x="0.138" z="4.5"/>
    <point x="0.418" z="7.0"/>
  </polygon>
</module>
```

Транспорт внутри подобного модуля самостоятельно отслеживает ситуации, когда частица может сразу же вернуться в объект из которого вылетела

Тестирование кода




```
TEST_CLASS(mcTransportRectanglePolygonSideHoleTest)
{
public:

	TEST_METHOD(getDistanceOutside)
	{
		double d;
		auto transport = createTestTransport();
		mcParticle p;
		// 1
		p.p = geomVector3D(-2,7,-10);
		p.u = geomVector3D(0,0,1);
		d = transport->getDistanceOutside(p);
		Assert::AreEqual(DBL_MAX, d, DBL_EPSILON*10, L"getDistanceOutside failed", LINE_INFO());
		// 2
		p.p = geomVector3D(0,12,-10);
		p.u = geomVector3D(0,0,1);
		d = transport->getDistanceOutside(p);
		Assert::AreEqual(10.0, d, DBL_EPSILON * 10, L"getDistanceOutside failed", LINE_INFO());
		// 3
		p.p = geomVector3D(-7,0,15);
		p.u = geomVector3D(0,0,-1);
		d = transport->getDistanceOutside(p);
		Assert::AreEqual(8.0, d, DBL_EPSILON * 10, L"getDistanceOutside failed", LINE_INFO());
		// 4
		p.p = geomVector3D(2,5,5);
		p.u = geomVector3D(0,0,1);
		d = transport->getDistanceOutside(p);
		Assert::AreEqual(DBL_MAX, d, DBL_EPSILON * 10, L"getDistanceOutside failed", LINE_INFO());
		...
```

Стандартные средства тестирования Visual Studio удобны и как средство постоянного контроль кода и как средство отладки в процессе разработки. Всегда можно поставить точку прерывания в месте интереса и посмотреть внутри.

Demo

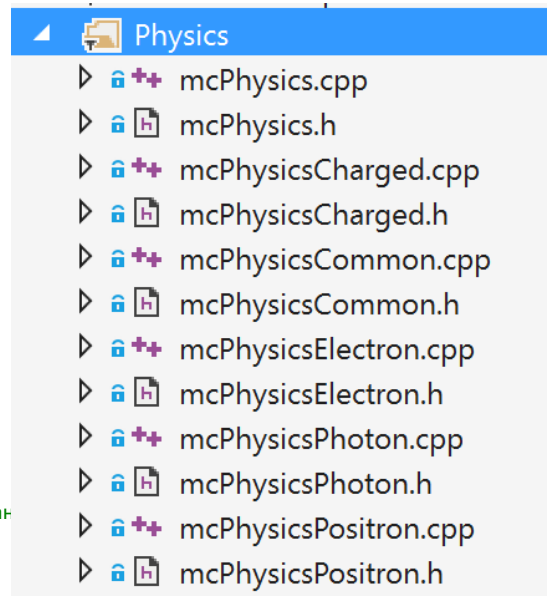
- Тестирование модулей кода

Физика взаимодействия излучения с веществом

МС позволяет легко расширеть транспорт на новые типы частиц

Базовый класс моделирования физических процессов

```
class mcPhysics
{
public:
        mcPhysics(void);
        virtual ~mcPhysics(void);

        // Расчет средней длины пробега в среде.
        virtual double MeanFreePath(double ke, const mcMedium& med, double dens) const = 0;

        // Разыгрывание акта взаимодействия.
        // Подразумевается, что указатель на часицу является указателем на элемент массива (стэка).
        // Если в результате взаимодействия вместо одной частицы образуется несколько, лишние
        // поместятся в стэк на основании указанного адреса текущего курсора.
        // Возвращается выделившаяся в точке в результате взаимодействия энергия.
        virtual double DoInterruction(mcParticle* p, const mcMedium* med) const = 0;

        // Перемещение частицы на заданное расстояние. Возвращает переданную на пути энергию.
        // Для фотонов это просто изменение координаты частицы.
        // В переменной step возвращается реальный шаг. Для заряженных частиц он будет отличаться от задан
        // если шаг моделирования непрерывных потерь окажется меньше.
        // В последнем случае транспорт определяет, что дискретное событие разыгрывать не следует.
        virtual double TakeOneStep(mcParticle* p, const mcMedium& med, double& step) const = 0;

        // Проверка частицы на предмет энергии ниже критической и при необходимости ее уничтожение.
        // Выделяющаяся энергия помещается в edep.
        // Уничтожение именно здесь, так как возможны процессы типа аннигиляции, что известно только физике.
        virtual bool Discarge(mcParticle* p, const mcMedium& med, double& edep) const = 0;

        // Physics utilities
        static void GetRandomPhi(double rnum, double* cosPhi, double* sinPhi);
        static void ChangeDirection(double cosTheta, double sinTheta, double cosPhi, double sinPhi, geomVector3D& u);
        static void GoInRandomDirection(double rnum1, double rnum2, geomVector3D& u);

        static mcParticle* DuplicateParticle(mcParticle* p);
        static void DiscardParticle(mcParticle* p);
};
```



Класс физики определяет интерфейс, которому должен удовлетворять класс взаимодействий частиц. МС позволяет легко расширять транспорт на любые типы частиц. Для этого нужно наследовать класс физики и дополнить информацией о сечениях.

Класс транспорта фотонов (mcPhysicsPhoton)

```
class mcPhysicsPhoton : public mcPhysics
{
public:
        mcPhysicsPhoton(void);
        virtual ~mcPhysicsPhoton(void);

        double MeanFreePath(double ke, const mcMedium& med, double dens) const override;
        double DoInterruction(mcParticle* p, const mcMedium* med) const override;
        double TakeOneStep(mcParticle* p, const mcMedium& med, double& step) const override;
        bool Discarge(mcParticle* p, const mcMedium& med, double& edep) const override;

protected:
        static void ProducePair(mcParticle* p, const mcMediumXE* pMedium);
        static void GetPairEnergies(mcRng& rng, const mcMediumXE* pMedium, double e_in, double* ke_elec1, double* ke_elec2);
        static void GetPairAngle(mcRng& rng, const mcMediumXE* pMedium, double eScaled_in, double ke_elec, double* cosTheta, double* sinTheta);
        static double PairRejectionFunction(double x, double zFactor, double r, double oneOverR, double val1);

        static void ComptonScatter(mcParticle* p, const mcMediumXE* pMedium);
        static void GetComptonSplit(mcRng& rng, double e_in,
        double* e_phot, double* cosTheta_phot, double* sinTheta_phot,
        double* ke_elec, double* cosTheta_elec, double* sinTheta_elec);

        static double DoPhotoelectric(mcParticle* p, const mcMediumXE* pMedium);
        static void GetPhotoelectricEnergies(double rnum, const mcMediumXE* pMedium, double e_in, double* e_dep, double* ke_elec, double* e_fluor);
        static void ChangePhotoelectronDirection(mcRng& rng, double ke_elec, geomVector3D& u);

        static void RayleighScatter(mcParticle* p, const mcMediumXE* pMedium);
        static void GetRayleighAngle(mcRng& rng, const mcMediumXE* pMedium, double e_phot, double* cosTheta, double* sinTheta);
};
```

Класс транспорта фотонов наследует класс физики и переопределяет виртуальные функции. Собственно физика взаимодействий скрыта внутри и не подразумевает знаний об окружающем коде за исключением библиотеки сечений.

Обработка взаимодействия фотона (DoInterruction)

```
double mcPhysicsPhoton::DoInterruction(mcParticle* p, const mcMedium* med) const
{
	double e_dep = 0; // энергия, выделившаяся в результате взаимодействия (фото-эффект)

	const mcMediumXE* m = (const mcMediumXE*)med;

	double logKE = log(p->ke);
	int iLogKE = (int)(m->iLogKE0_phot + logKE * m->iLogKE1_phot);
	mcRng& rng = p->thread_->rng();

	double rayleighFactor = 1;
	if (m->rayleigh)
		rayleighFactor = m->raylFactor0[iLogKE] + logKE * m->raylFactor1[iLogKE];

	if (m->rayleigh && rng.rnd() < 1.0 - rayleighFactor)
		RayleighScatter(p, m);
	else
	{
		double branch = rng.rnd();

		// br1 = Pr(pair production)
		if (branch < m->br10_phot[iLogKE] + logKE * m->br11_phot[iLogKE])
			ProducePair(p, m);

		// br2 = Pr(pair production) + Pr(Compton)
		else if (branch < m->br20_phot[iLogKE] + logKE*m->br21_phot[iLogKE])
			ComptonScatter(p, m);
		else
			e_dep = DoPhotoelectric(p, m);
	}

	return e_dep;
}
```

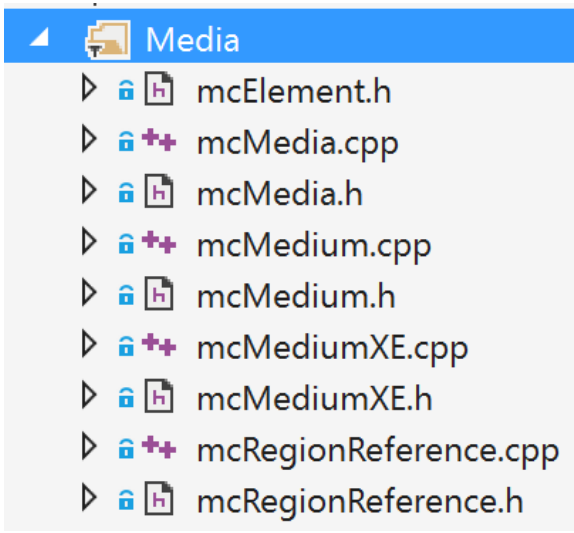
При разыгрывании типа взаимодействия из библиотеки сечений берется относительная вероятность каждого типа взаимодействия в конкретной среде для конкретной энергии частицы. Затем обрабатывается конкретный тип.

Среды, сечения, PEGS4

МС воспроизводит исключитиельно физику и сечения EGS4 / PEGS4

Базовый класс описания среды (mcMedium)

```
class mcMedium
{
public:
	mcMedium(void);
	virtual ~mcMedium(void);

	enum STATUS { EMPTY, LOADED, FAILED };

	virtual void read(istream& is) = 0;

protected:
	// Функция извлекает 2 слова из строки.
	// Первое слово сравнивается с именем параметра.
	// При несовпадении устанавливается исключение.
	// По окончании сторка line содержит остаток строки.
	string ParseLine(string& line, const char* param);

public:
	// General:
	STATUS status_;		// Текущее состояние данных
	string name_;		// Name of medium
	double density_;	// Default density of medium
	vector<mcElement> elements_;
};
```



Организационно класс описания среды помимо сечений и предобработанных данных содержит ссылку на класс описания физики транспорта в ней.

Библиотека описания сред (mcMedia)

```
class mcMedia
{
public:
        // Добавление сред по названиям.
        // Подготовка сечений происходит после перечисления сред вызовом
        // функций инициализации из потока или файлов, отдельно по группам типов частиц.
        void addName(const char* mname);

        // Поиск среды по имени, полезный для инициализации транспортов
        short getMediumIdx(const char* mname) const;

        const mcMediumXE* getMediumXE(short idx) const;

        void initXEFromStream(istream&);
        void initXEFromFile(const string& fname);

        // Возвращает указатель объекта физических расчетов для частицы указанного типа
        const mcPhysics* getPhysics(int ptype) const;

        // Возвращает указатель объекта физических расчетов для частицы указанного типа
        const mcMedium* getMedium(int ptype, int idx) const;

        vector<mcMedium*> Media() { return xes_; }

protected:
        // Список имен сред
        vector<string> mnames_;

        // Параметры сред транспорта фотонов и электронов
        vector<mcMedium*> xes_;

        vector<mcPhysics*> physics_;
};
```

Это контейнер, из которого извлекаются конкретные экземпляры класов описания конкретных сред в зависимости типов частицы и среды.

Описание среды для электронного и фотонного транспорта

```
class mcMediumXE : public mcMedium
{
public:
	mcMediumXE(void);
	virtual ~mcMediumXE(void);

	void read(istream& is) override;

protected:
	static void FixStepSize(int nBins_elec, double eStep,
	double iLogKE0, double iLogKE1,
	vector<double>& stepSize0, vector<double>& stepSize1,
	vector<double>& dedx0, vector<double>& dedx1);

	void InitializeAngularDistribution();

public:
	// Photons:
	// Log energy index:
	double iLogKE0_phot;
	double iLogKE1_phot;

	// Interactions:
	double eventCutoff_phot;
	vector<double> br10_phot;		// Branching ratio 1
	vector<double> br11_phot;
	vector<double> br20_phot;		// Branching ratio 2
	vector<double> br21_phot;
	int rayleigh;				// Switch to turn on Rayleigh scattering
	...
```

Текущая версия читает стандартный PEGS4 файл.

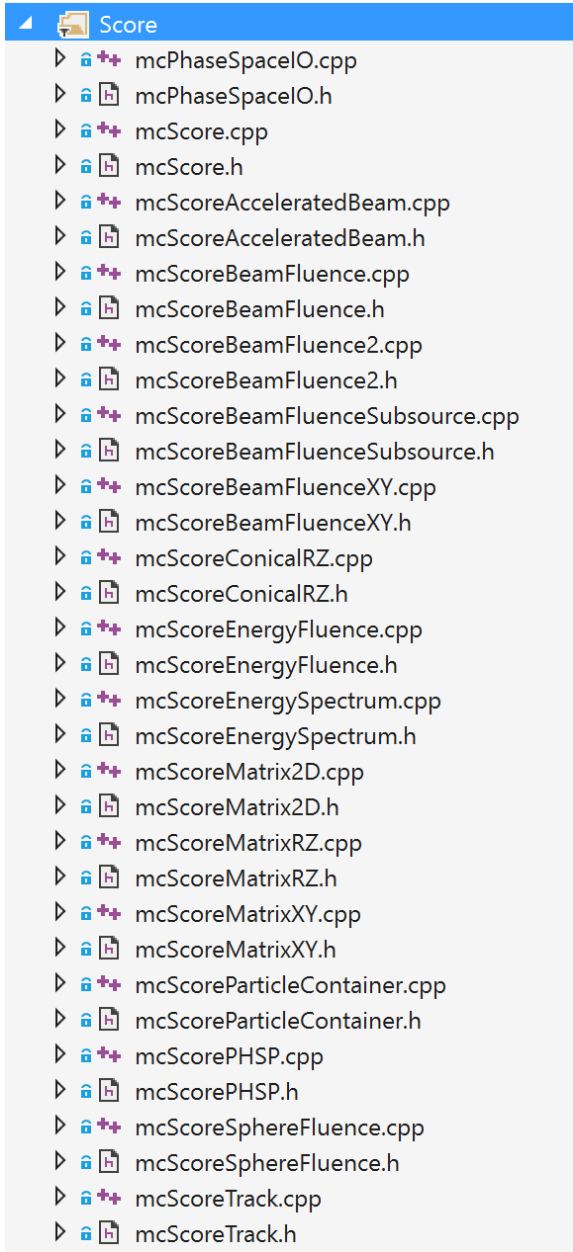
Сбор статистики (Scoring)

Класс статистики отслеживают события и учитывают внутри в зависимости от задачи. Это могут быть матрицы накопления выделившийся энергии, треки, спектры, биологически эквивалентные дозы и т.д.

Базовый класс сбора статистики (mcScore)

```
class mcScore : public mcObj
{
public:
        mcScore(const char* module_name, int nThreads);
        virtual ~mcScore(void);

        virtual void ScoreFluence(const mcParticle& particle) { }

        virtual void ScorePoint(double edep
                      , int iThread
                      , const mcRegionReference& region
                      , mc_particle_t pt
                      , const geomVector3D& p0) {
                etotal_[iThread] += edep;
        }

        virtual void ScoreLine(double edep
                      , int iThread
                      , const mcRegionReference& region
                      , mc_particle_t pt
                      , const geomVector3D& p0
                      , const geomVector3D& p1) {
                etotal_[iThread] += edep;
        }

        // Вывод изображения детектора в формате VRML.
        // Матрица преобразования координат в мировую систему необходима,
        // так как scoring задан в системе координат объекта транспорта, ..
         с которым он связан.
        virtual void dumpVRML(ostream&) const;

        // Вывод содержимого перемещен не в оператор, а в функцию,
        // так как требуется поддержка наследования.
        virtual void dumpStatistic(ostream&) const;
        ...
```

Класс определяет интерфейс классов, реагирующих на события транспорта и обрабатывающих поступающую информацию по своему усмотрению.



Пример класса сбора информации о потоке излучения

```
void mcScoreEnergyFluence::ScoreFluence(const mcParticle& particle)
{
      if (particle.t == pt_)
      {
            int iThread = particle.thread_->id();
            double edep = particle.ke * particle.weight;
            etotal_[iThread] += edep;

            // !!! Scoring вызывается до перемещения частицы на поверхность
            // объекта к которому привязана частица.
            // Нужно ее перенести здесь на поверхность.
            geomVector3D p = particle.p + (particle.u * (-particle.p.z() / particle.u.z()));

            double r = p.lengthXY();
            int ridx = int(r / rstep_);
            if (ridx < nr_)
                  fluence_[iThread][ridx] += edep;
      }
}
```

Все что нужно для наблюдения за симуляцией – это реагирование на события. Пользователь волен делать с поступающей информацией все что угодно. Накапливать выделяюшуюся в ячейках пространства энергию, сохранять траектории частиц и т.д.

Пример работы с треками

```
void mcScoreTrack::score(int iThread, ...)
{
        ...
        geomVector3D v = pp1 - pp0;
        v.normalize();
        double d = mcGeometry::getDistanceToInfiniteCylinderInside(pp0, v, R_);
        pp1 = pp0 + (v * d);
        ...
        if (pt == MCP_PHOTON) {
                photons_[iThread].push_back(pp0);
                photons_[iThread].push_back(pp1);
        }
        else if (pt == MCP_NEGATRON) {
                electrons_[iThread].push_back(pp0);
                electrons_[iThread].push_back(pp1);
        }
        else if (pt == MCP_POSITRON) {
                positrons_[iThread].push_back(pp0);
                positrons_[iThread].push_back(pp1);
        }
        ...
}

void mcScoreTrack::dumpVRML(ostream& os) const
{
        int i, j, ia;

        os << "# Particles tracks: " << this->getName() << endl;
        os << "Group {" << endl;
        os << "  children [" << endl;

        for (ia = 0; ia < 4; ia++)
        {
                const vector<vector<geomVector3D>>& t = ia == 0 ? photons_ : ia == 1 ? electrons_ : positrons_;
                os << "    Transform {" << endl;
                os << "      children Shape {" << endl;
                ...

}
```

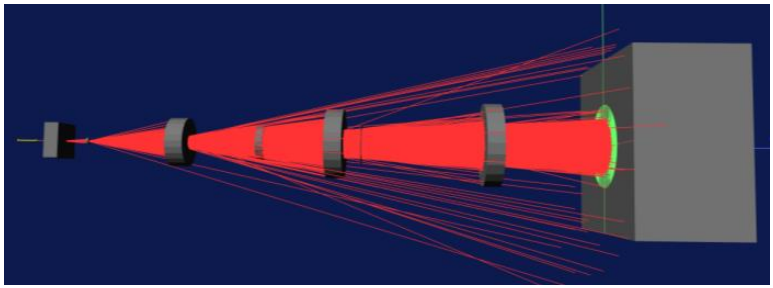


Рис. 1 *Транспорт излучения в системе формирования терапевтических полей облучения в режиме отображения треков частиц. Зеленый объект внутри фантома изображает дозовую матрицу в геометрии.*

Данный пример реализует простой сценарий. Каждый раз, когда частица перемещается на какое-то расстояние, скоритг сохраняет тип частицы и отрезок в пространстве.
По окончании симуляции отрезки выводятся в графический файл.

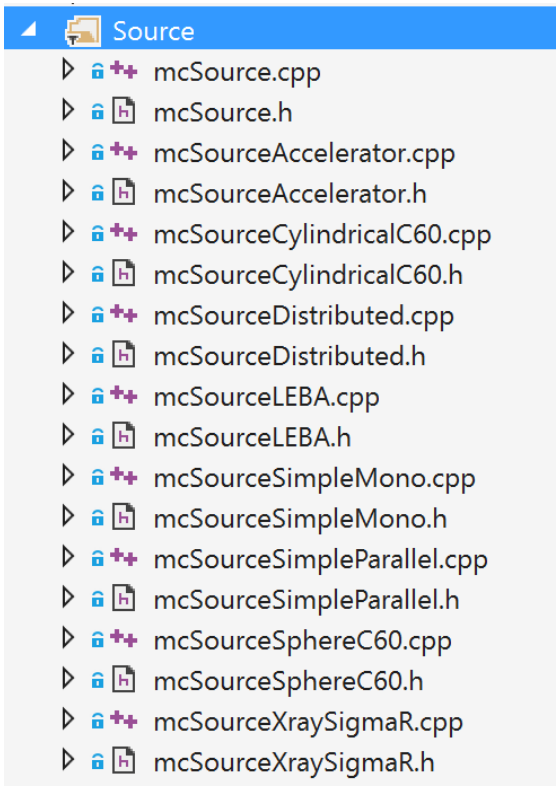
Источник частиц (Source)

Базовый класс источника определяет интерфейс, который должен поддерживать пользовательский источник излучения.

Код MC содержит большой набор готовых классов.

Базовый класс источника излучения (mcSource)

```
class mcSource : public mcObj
{
      public:
      mcSource();
      mcSource(const char* name, int nThreads);
      virtual ~mcSource();

      void setScoreTrack(double R, double Z1, double Z2, double EMIN,
           bool doPhotons, bool  doElectrons, bool doPositrons);
      bool IsGamma() const { return isGamma_; };

      void setModuleName(const char* s);
      const char* getModuleName() const { return attached_module_; }

      virtual void sample(mcParticle& p, mcThread* thread) = 0;

      double etotal() const;

      virtual void dumpVRML(ostream& os) const;
};
```



Главная цель этой группы файлов – ответить на вопрос **sample()**, т.е. выдать очередную случайную частицу.

Пример источника аппарата C-60 (mcSourceCylindricalC60)

```
void mcSourceCylindricalC60::sample(mcParticle& p, mcThread* thread)
{
	mcRng& rng = thread->rng();

	p.t = MCP_PHOTON;
	p.q = 0;
	p.ke = rng.rnd() < 0.5 ? 1.33 : 1.17;

	double z = h_ * rng.rnd();
	double r = r_ * sqrt(rng.rnd());
	double s, c;
	mcPhysics::GetRandomPhi(rng.rnd(), &c, &s);
	p.p.set(p_.x() + r * c, p_.y() + r * s, p_.z() + z);
	p.plast = p.p;

	mcPhysics::GoInRandomDirection(rng.rnd(), rng.rnd(), p.u);

	p.weight = 1;
	p.thread_ = thread;
	p.trackScore_ = trackScore_;
	etotal_[thread->id()] += p.ke;
}
```

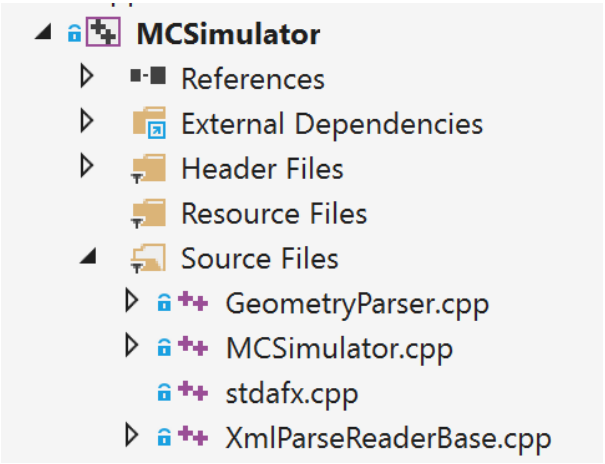
Случайным образом разыгрывается место появления фотона при условии равномерного распределения в цилиндре и для него случайным образом задается направление движения.

MCSimulator Reference Implementation

Готовая исполняемая программа MCSimulator позволяет решать значительную часть практических задач вообще без программирования. Задача может полностью описываться двумя XML файлами. Вместе с тем, она является справочным примером использования библиотек MC.

Демонстрационное приложение (MCSimulator)

```
int _tmain(int argc, _TCHAR* argv[])
{
	if (argc != 3)
	{
		wcout << L"Usage:" << endl;
		wcout << argv[0] << L" params.xml geometry.xml" << endl;
		return -1;
	}

	int i;
	int nThreads = concurrency::GetProcessorCount();
	//int nThreads = 1;

	double simStartTime = TIME;

	mcThread* threads = new mcThread[nThreads];
	for (i = 0; i < nThreads; i++)
	threads[i].setId(i);

	wcout << L"nThreads =  " << nThreads << endl;
	try
	{
		// Используемые среды
		mcMedia media;
		media.addName("H2O700ICRU");
		media.addName("AIR700ICRU");
		media.addName("LUNG700ICRU");
		…….
```



Это полноценное консольное приложение, демонстрирующее весь набор функциональности библиотеки MC.

Парсер демонстрационного приложения (GeometryParser)

```
class GeometryParser
{
public:
	static enum mc_particle_t convert_S2T_ptype(const wchar_t* st);
	static enum spectrum_distr_t convert_spec_type(const wchar_t* st);
	static enum profile_distr_t convert_profile_distr_type(const wchar_t* st);

	static mcTransport* ParseTransport(const XPRNode& geometry, const mcMedia* media, int nThreads);
	static mcScore* ParseScore(const XPRNode& item, int nThreads);
	static mcSource* ParseSource(const XPRNode& item, int nThreads);
};
…….

mcTransport* GeometryParser::ParseTransport(const XPRNode& geometry, const mcMedia* media, int nThreads)
{
…….

	//
	// Создание объектов
	//
	if (_wcsicmp(geomType.c_str(), L"cylinder") == 0)
	{
	t = new mcTransportCylinder(geomVector3D(x0, y0, z0), geomVector3D(vx, vy, vz), geomVector3D(xx, xy, xz), radius, height);
	}
	else if (_wcsicmp(geomType.c_str(), L"cone") == 0)
	{
	t = new mcTransportCone(geomVector3D(x0, y0, z0), geomVector3D(vx, vy, vz), geomVector3D(xx, xy, xz), radius, height);
	}

…….
return t;
}
```

Парсер сканирует XML файл задачи и программно формирует дерево экземпляров классов для ее решения.

XML файл описания геометрии

```
<accelerator>
  <!--Tungsten target-->
  <module type="cylinder" name="Target" medium="W700ICRU" density="1">
    <Color r="0" g="0.5" b="1" t="0.2" />
    <position unit="cm" x="0" y="0" z="0" />
    <normal x="0" y="0" z="1" />
    <xaxis x="1" y="0" z="0" />
    <size unit="cm" radius="1.0" height="0.2"/>
  </module>

  <module type="planefilter" name="Trap" medium="AIR700ICRU" density="1">
    <Color r="0" g="0.5" b="0.5" t="0.8" />
    <position unit="cm" x="0" y="0" z="0.2" />
    <normal x="0" y="0" z="1" />
    <xaxis x="1" y="0" z="0" />
  </module>

</accelerator>
```

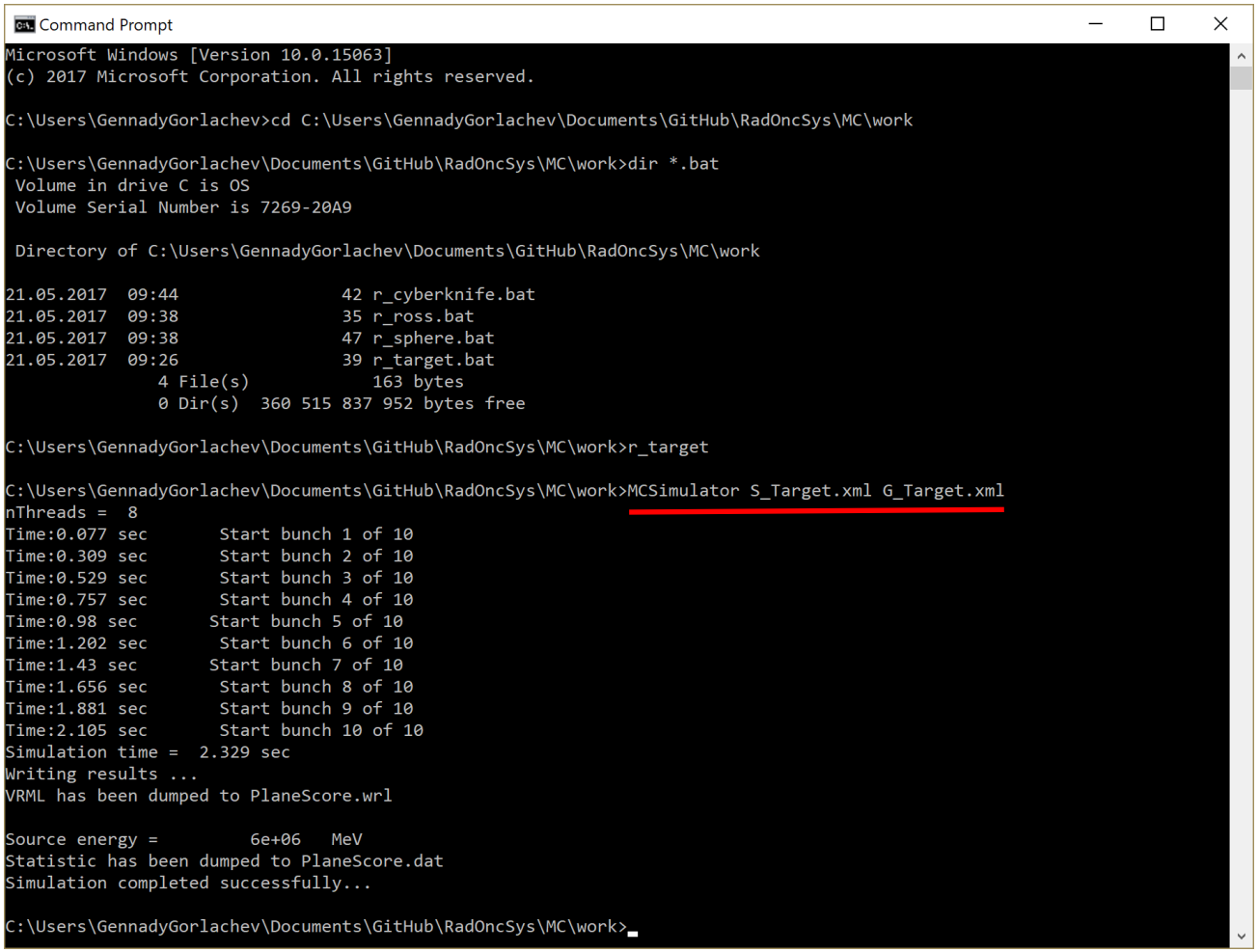
Простейший файл геометрии симуляции тормозного излучения после вольфрамовой мишени.
Сцена состоит из двух модулей: цилиндра мишени и плоскости захвата частиц.

XML файл описания задачи

```
<?xml version="1.0" encoding="utf-8"?>

<!--
Bremsstrahlung production by electron beam on the target.
-->

<input>
  <simulation nhistories="100000" nbanches="10">
  </simulation>

  <options>
    <vrmlfile>PlaneScore.wrl</vrmlfile>
    <statfile>PlaneScore.dat</statfile>
    <transCutoff_elec unit="MeV" ecat ="0.0"/>
  </options>

  <source name="Electron beam" module="Target" trackparticles="false">
    <radiation type="electron" energy="6.0" />
    <shape direction="conical" size="0.00" angle="0"/>
    <position unit="cm" x="0" y="0" z="-1.0" />
    <direction x="0" y="0" z="1" />
  </source>

  <score type="fluence_plane" module="Trap" pt ="photon"/>

</input>
```

Файл описания задачи задает количество симуляций, указывает файлы вывода результатов, тип и параметры источника, классы сбора статистики.

Запуск симуляции



Симулятор – обычное консольное приложение.

В качестве аргументов ему нужно указать два файла описания задачи и геометрии.

Demo

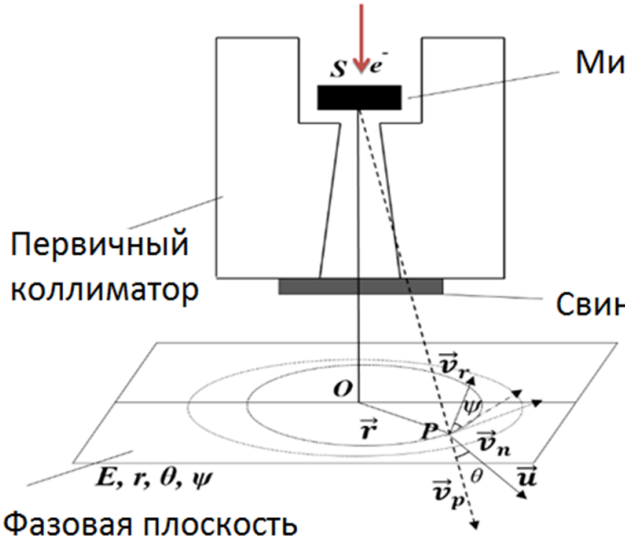
- Структура демонстрационного приложения
- Запуск симуляции
- Обзор графического файла

Пример симуляции Киберножа

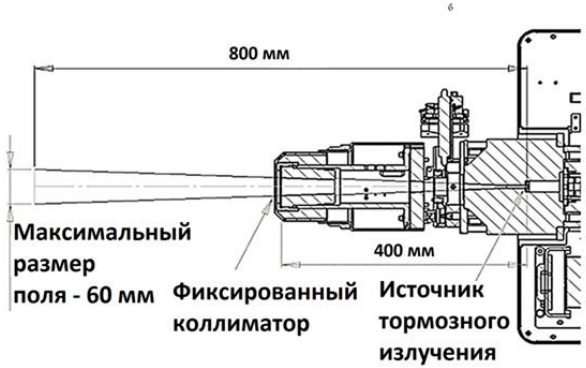
От понимания свойств потоков коллимированного излучения до модели источника излучения

Модель источника медицинского ускорителя. Верификация модели на примере CyberKnife

CyberKnife (НИИ нейрохирургии им. Н.Н. Бурденко)

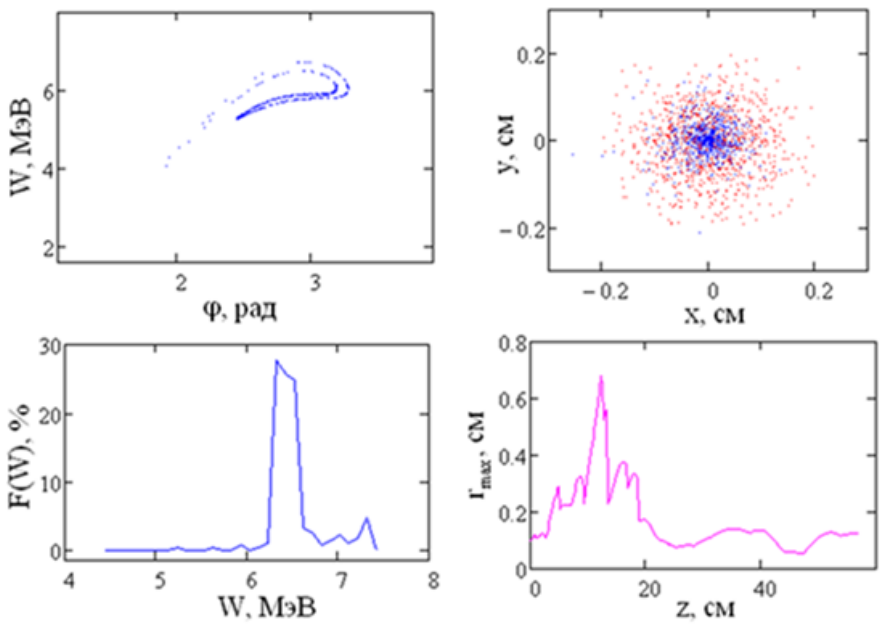


- Радиальная симметрия
- 4 параметра: ***E, r, θ , ψ***

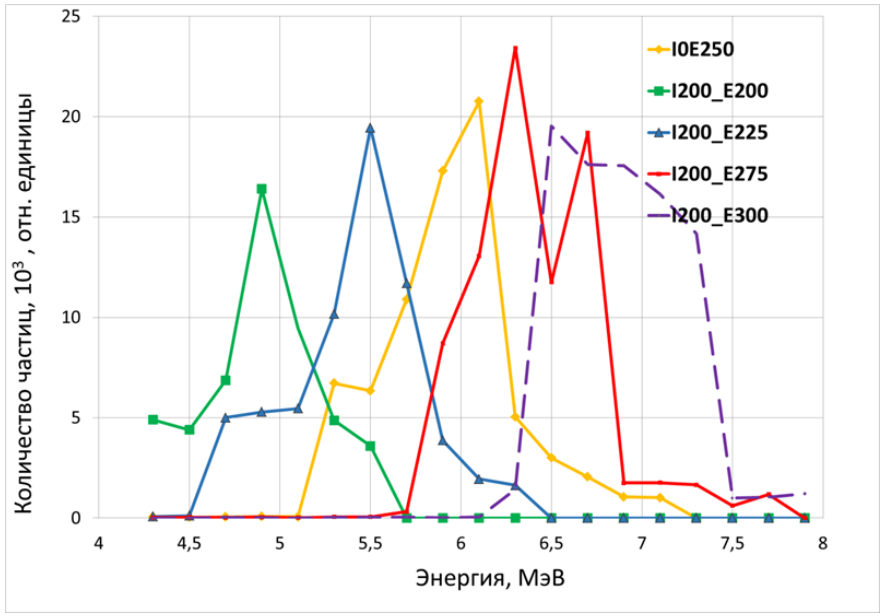


Формирование гистограмм распределений частиц

Влияние характеристик электронного пучка на мишени на свойства дозовых распределений

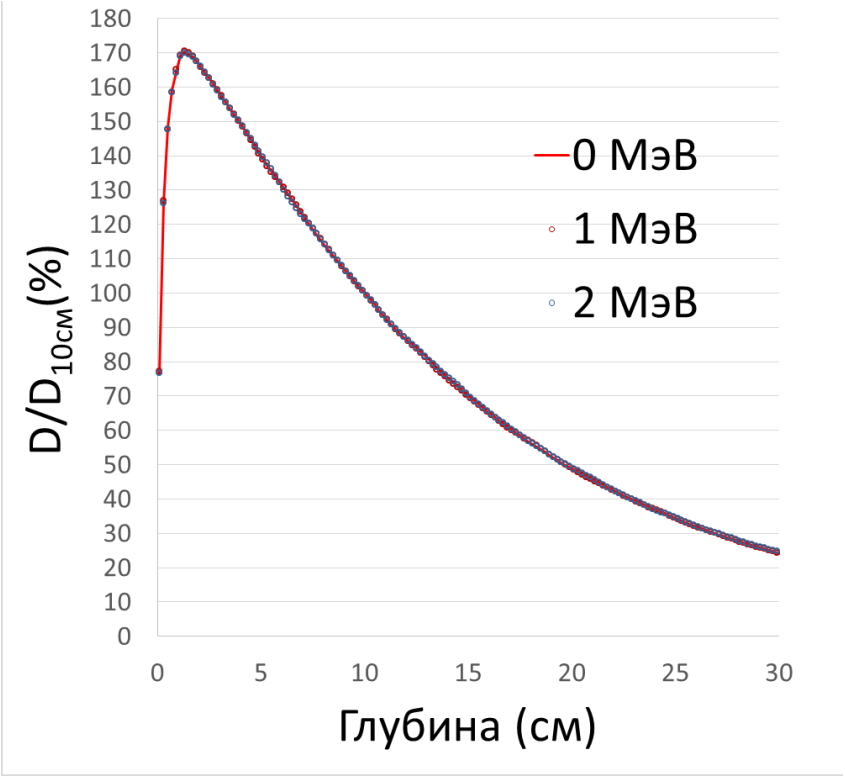


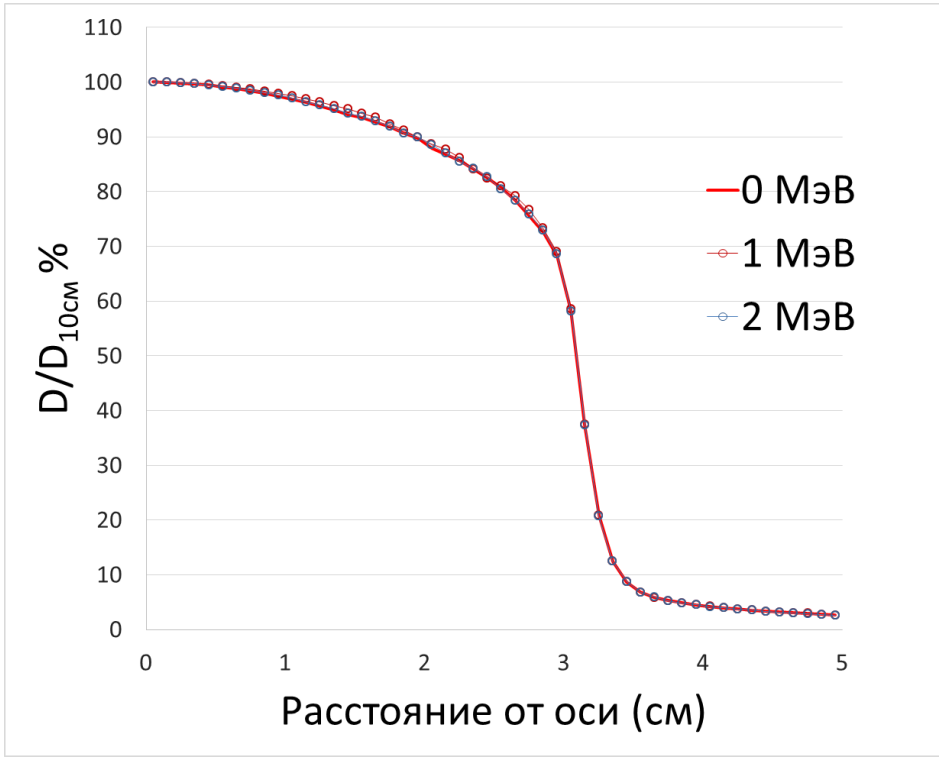
Радиофизическое моделирование
для точного описания ускоренных частиц на радиационной мишени



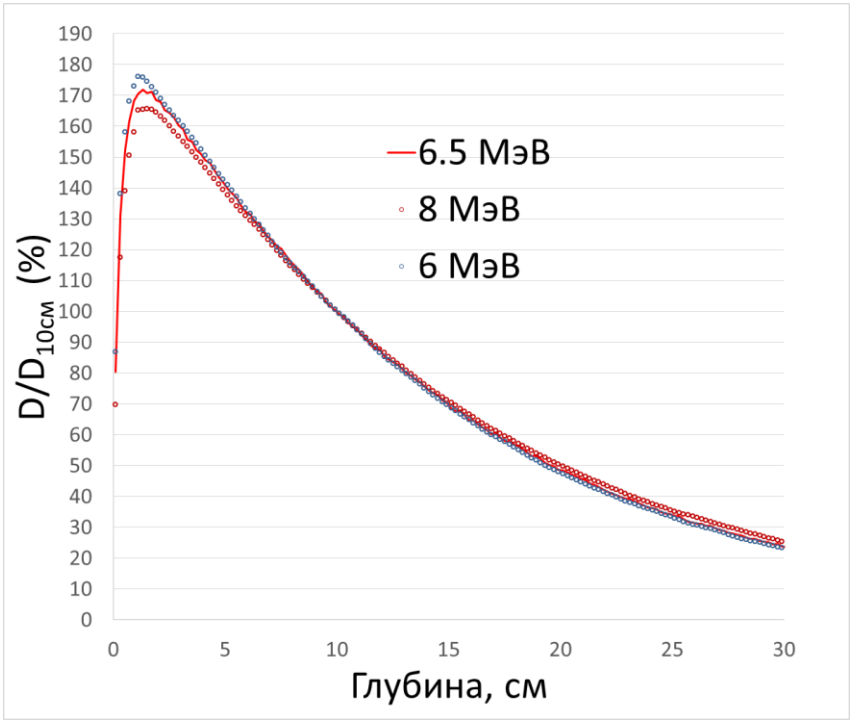
Графики спектров представлены при различных токах инжекции (I) и напряженностях ускоряющего поля (E)

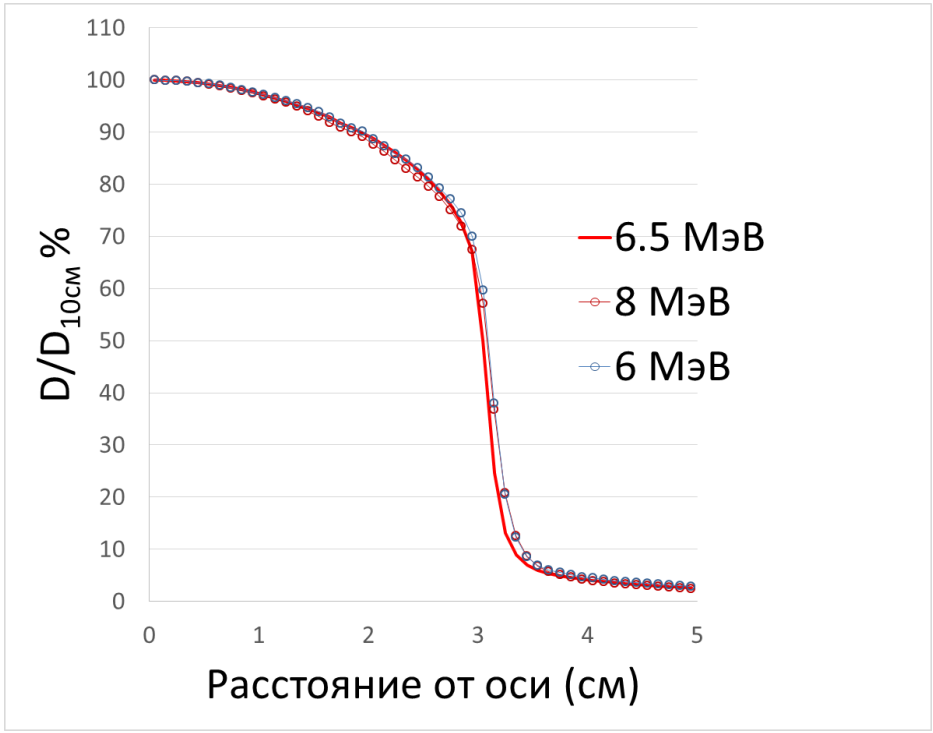
Влияние спектра электронного пучка на дозовые распределения

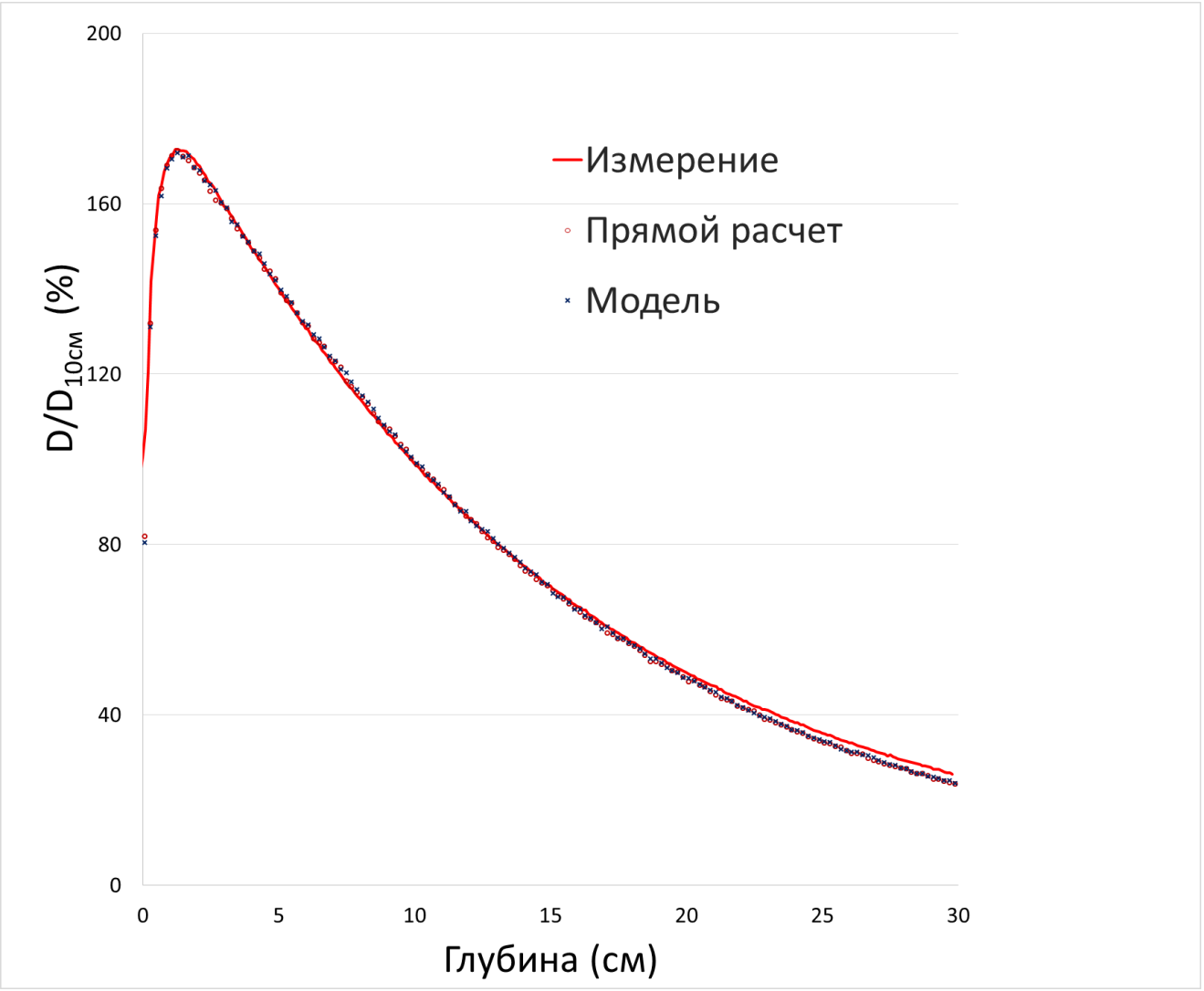




Влияние энергии электронного пучка на дозовые распределения



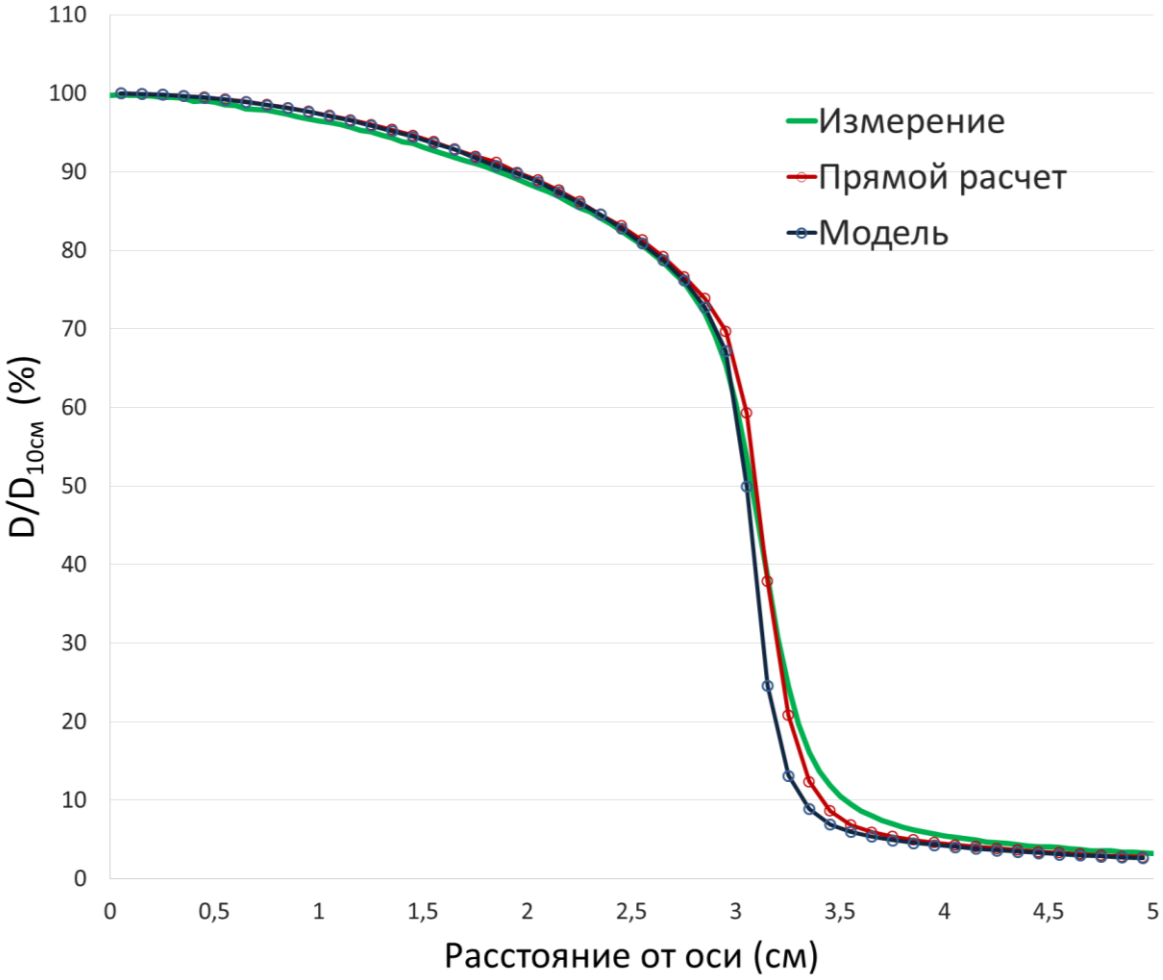




Воспроизведение экспериментальных данных:

- **1%** по отношению к максимальной дозе
- **5 %** на глубине 30 см (по отношению к $D_{30см}$)

Размер поля 60 мм. Энергия электронов 6.5 МэВ. Расстояние до фантома 70 см.

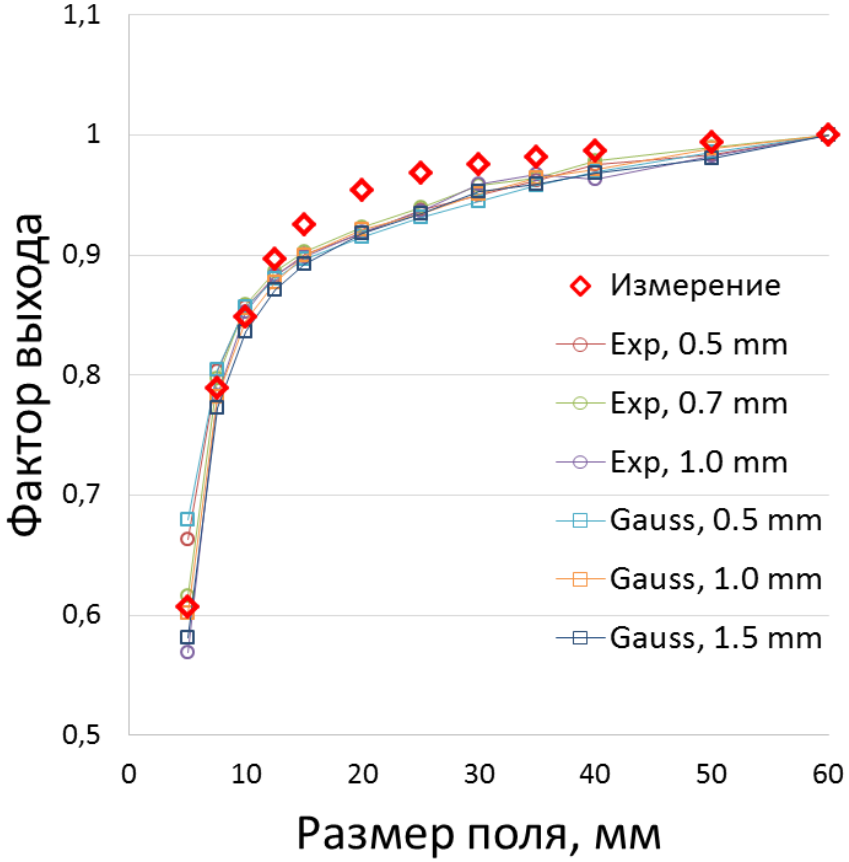


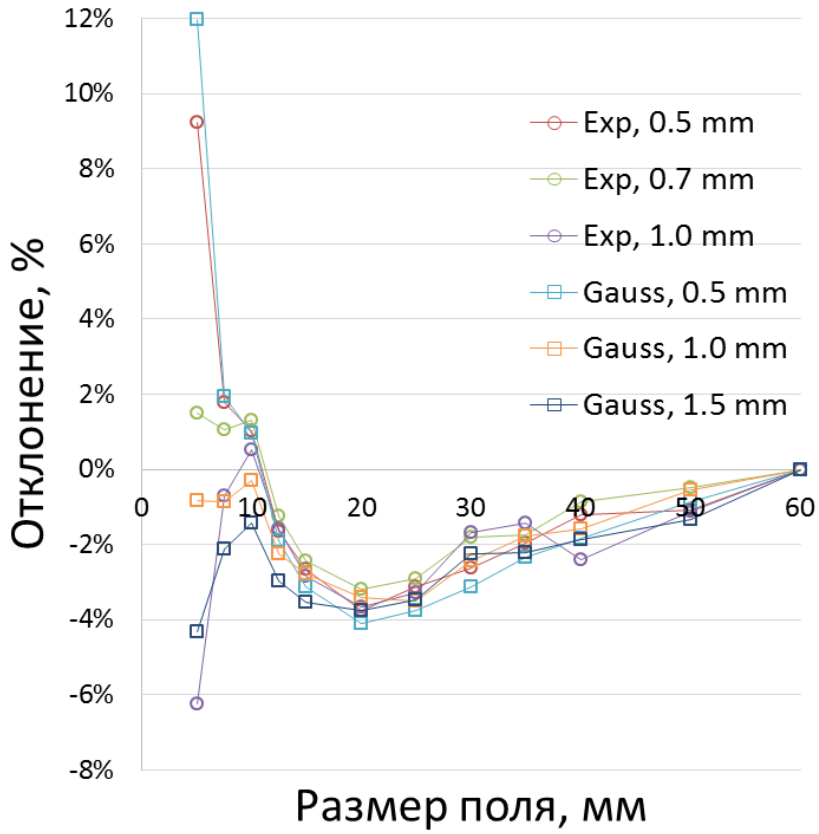
Воспроизведение экспериментальных данных:

- **0,5 мм** при прямом расчете
- **1 мм** при использовании модели

Размер поля 60 мм. Энергия электронов 6.5 МэВ. Расстояние до фантома 70 см.

Воспроизведение факторов выхода





Параметры и условия симуляции

- CK_6X.xml CK_before_model.xml MCSimulator. exe - для файла модели источника
- CK_sim_from_model.xml CK_after_model.xml MCSimulator. exe - дозовое распределение в фантоме из модели источника
- CK_6X.xml CyberKnife.xml MCSimulator. exe - полная симуляция дозовых распределений от электронного пучка на мишени до водного фантома

Параметры и условия симуляции

erKnife.xml | CK_before_model.xml | CK_after_model.xml | CK_sim_from_model.xml | CK_6X.xml

```
<!--
Источник может быть только один.
Возможны следующие типы источников, определяемые атрибутом "direction" параметра "shape".
Вриантs типов: simple, conical.
В коническом источнике задается радиус пучка в см и угл частиц к оси Z в градусах
-->

<!--
<source name="Cyberknife electron accelerator" trackparticles="false">
  <radiation type="electron" energy="6.0" />
  <shape direction="conical" size="0.00" angle="0"/>
  <position unit="cm" x="0" y="0" z="-80.3" />
  <direction x="0" y="0" z="1" />
</source>
-->

<!--
<source name="Cyberknife electron accelerator" trackparticles="false">
  <radiation type="electron" energy="6.5" />
  <shape direction="conical_sigmar" size="0.10" angle="0"/>
  <position unit="cm" x="0" y="0" z="-80.14" />
  <direction x="0" y="0" z="1" />
</source>
-->

<source name="Cyberknife RF source" module="Target" trackparticles="false">
  <!--Типы спектров spectrum="gauss/triangle/prism"-->
  <radiation type="electron" energy="6.0" ewidth="2.0" spectrum="prism" />
  <!--size = показатель экспоненциально спада по радиусу-->
  <shape direction="src_leba" size="0.05" />
  <position unit="cm" x="0" y="0" z="-80.14" />
  <direction x="0" y="0" z="1" />
</source>

<!--
Возможны следующие типы скоринга: "phsp", "rz", "fluence", "fluence2", "rz_conical".
Название модуля должно совпадать с названием геометрического модуля в файле геометрии.
-->

<!--Накопитель частиц для модели источника-->
<score type="phsp_concentrator" module="PHSP detector" outfile="ck_phsp_model_60_20_05.dat"
       isxray="true" particles="0" focus="31">
  <size unit="cm" ne="50" nr="20" nt="20" na="20"
        emax="6.0" rmax="2.0" tmax="0.5" />
</score>

<!--Дозовое распределение в веерной RZ геометрии -->
<!--<score type="rz_conical" module="Phantom">
  <size unit="cm" nr="50" nz="175" rmax="5.0" zmin="0.0" zmax="35.0" ziso="10.0" sad="80"/>
</score>-->

<!--<score type="2D" module="Phantom">
  <size unit="cm" nx="100" nz="175"
        x1="-5.0" x2="5.0"
        y1="-0.5" y2="0.5"
        z1="0.0" z2="35.0" />
</score>-->

</input>
```

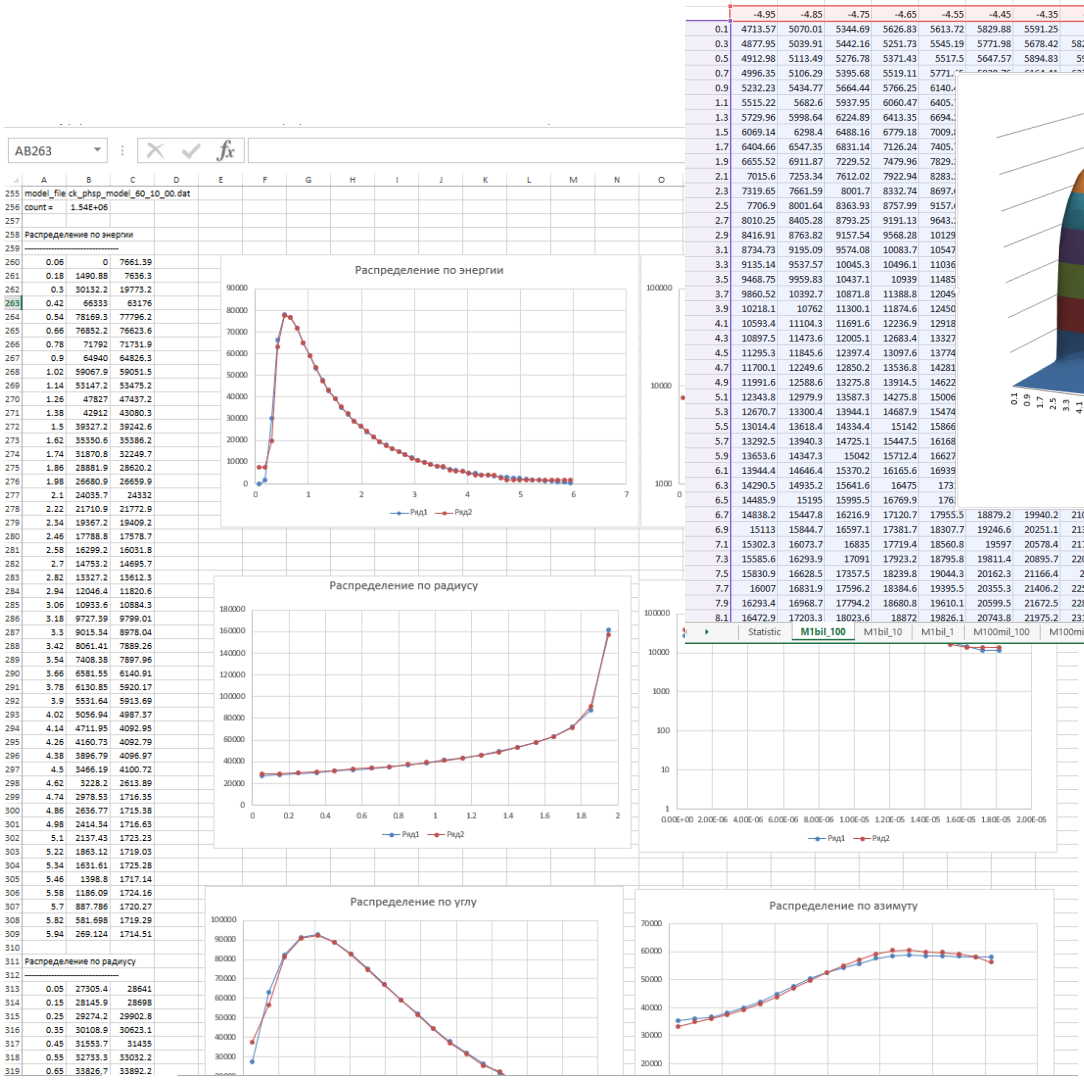
CyberKnife.xml | CK_before_model.xml | CK_after_model.xml | CK_sim_from_model.xml | CK_6X.xml

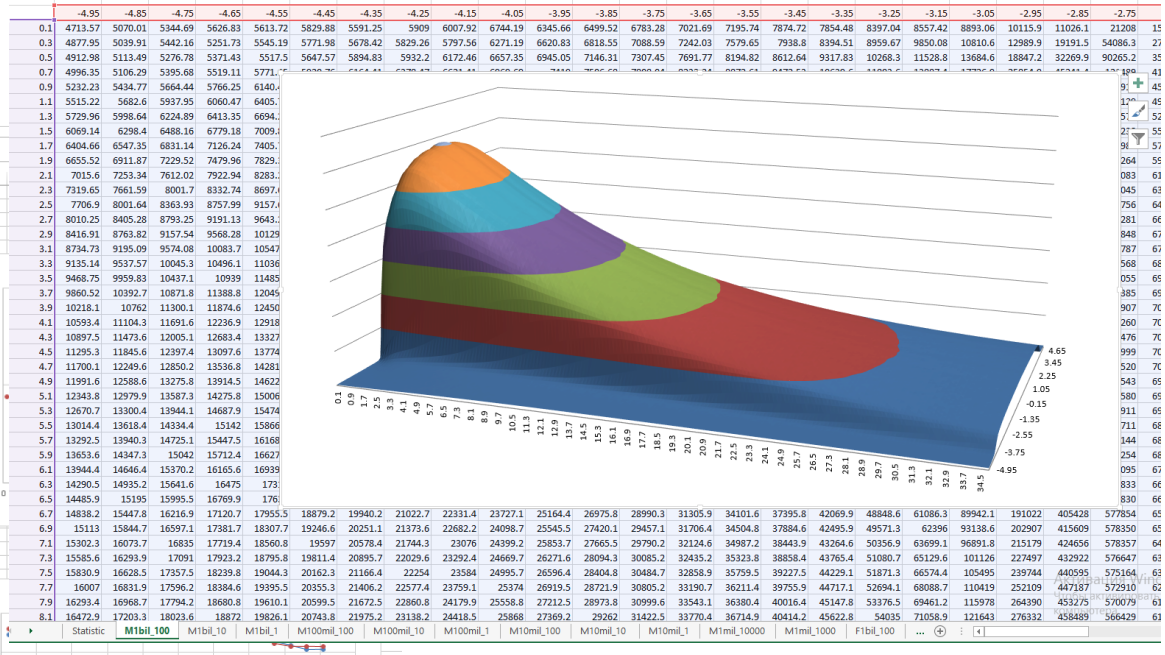
```
<?xml version="1.0" encoding="utf-8"?>

<accelerator>
  <!--Вольфрамовая мишень-->
  <module type="cylinder" name="Target" medium="W700ICRU" density="1">
    <Color r="0" g="0.5" b="1" t="0.2" />
    <position unit="cm" x="0" y="0" z="-80.14" />
    <normal x="0" y="0" z="1" />
    <xaxis x="1" y="0" z="0" />
    <size unit="cm" radius="0.95" height="0.14"/>
  </module>

  <!--<module type="axial_splitter" name="Splitter" medium="W700ICRU" density="1">
    <Color r="0.5" g="0.5" b="0" t="0.5" />
    <position unit="cm" x="0" y="0" z="-80" />
    <normal x="0" y="0" z="1" />
    <xaxis x="1" y="0" z="0" />
    <nsplit particle="photon" value="20"/>
  </module>-->

  <!--Первая часть аллюминиевой заглушки-->
  <module type="cylinder" name="AL1" medium="AL700ICRU" density="1">
    <Color r="1" g="1" b="1" t="0.5" />
    <position unit="cm" x="0" y="0" z="-78.6" />
    <normal x="0" y="0" z="1" />
    <xaxis x="1" y="0" z="0" />
    <size unit="cm" radius="0.635" height="0.1"/>
  </module>

  <!--Слой первичного коллиматора с аллюминиевым фильтром-->
  <module type="group" name="PRI_1">
    <position unit="cm" x="0" y="0" z="-78.5" />
    <normal x="0" y="0" z="1" />
    <xaxis x="1" y="0" z="0" />
    <module type="ring" name="PRI_1W" medium="W700ICRU" density="1">
      <Color r="0" g="0.5" b="0" t="0.5" />
      <position unit="cm" x="0" y="0" z="0" />
      <normal x="0" y="0" z="1" />
      <xaxis x="1" y="0" z="0" />
      <size unit="cm" r0="0.24" r1="8.25" height="3.1"/>
    </module>
    <module type="cylinder" name="PRI_1A" medium="AL700ICRU" density="1">
      <Color r="1" g="1" b="1" t="0.5" />
      <position unit="cm" x="0" y="0" z="0" />
      <normal x="0" y="0" z="1" />
      <xaxis x="1" y="0" z="0" />
      <size unit="cm" radius="0.24" height="0.85"/>
    </module>
  </module>

  <!--Коническая часть первичного коллиматора-->
  <module type="conicalhole" name="PRI_3" medium="W700ICRU" density="1">
    <Color r="0" g="0.5" b="0" t="0.5" />
    <position unit="cm" x="0" y="0" z="-68.6" />
    <normal x="0" y="0" z="-1" />
    <xaxis x="1" y="0" z="0" />
    <size unit="cm" r0="0.53" r1="8.25" height="6.8" focus="12.4"/>
  </module>
```

Типы «скоринга»

```
Возможны следующие типы скоринга: "phsp", "rz", "fluence", "fluence2", "rz_conical".
 Название модуля должно совпадать с названием геометрического модуля в файле геометрии.
 -->

 <!--Накопитель частиц для модели источника-->
 <score type="phsp_concentrator" module="PHSP detector"
outfile="ck_phsp_model_60_20_05.dat"
        isxray="true" particles="0" focus="31">
   <size unit="cm" ne="50" nr="20" nt="20" na="20"
         emax="6.0" rmax="2.0" tmax="0.5" />
 </score>

 <!--Дозовое распределение в веерной RZ геометрии -->
 <!--<score type="rz_conical" module="Phantom">
   <size unit="cm" nr="50" nz="175" rmax="5.0" zmin="0.0" zmax="35.0" ziso="10.0"
sad="80"/>
 </score>-->

 <!--<score type="2D" module="Phantom">
   <size unit="cm" nx="100" nz="175"
```

Statistic.dat → Excel





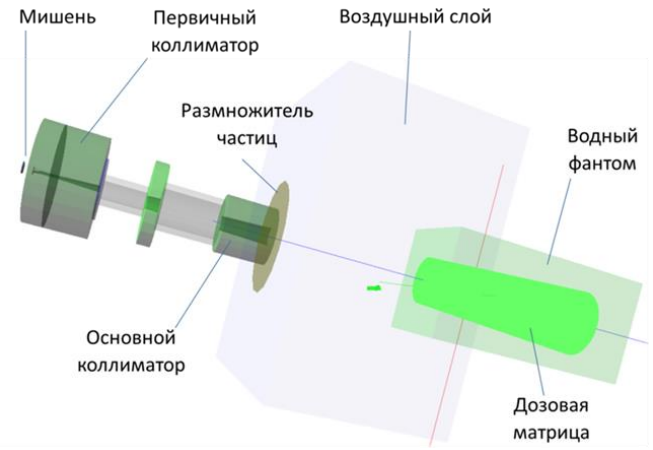
Дозовые распределения

Гистограммы распределений частиц для модели

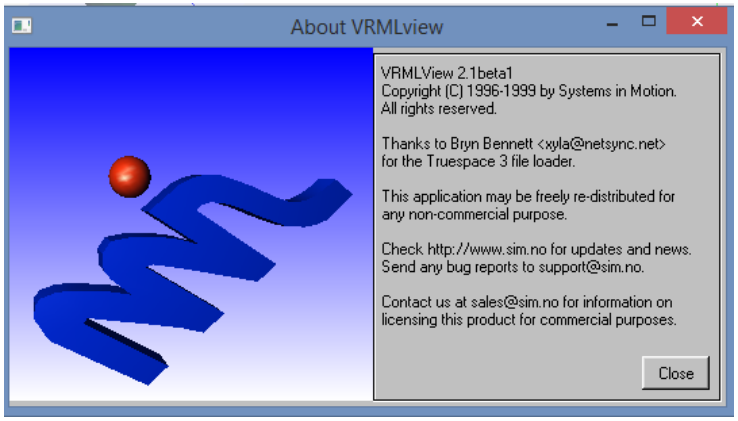
Vrml Viewer

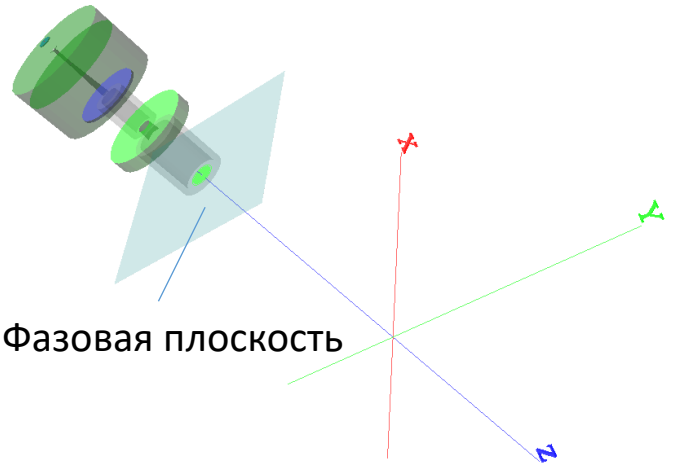
<vrmlfile>CyberKnife.wrl</vrmlfile>

<statfile>statistic.dat</statfile>



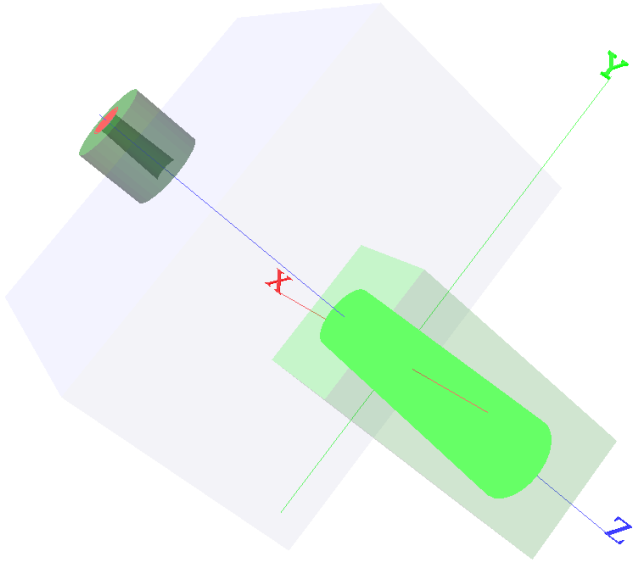
Общая сцена симуляции





Фазовая плоскость

Сцена симуляции до модели источника



Из модели источника

Симуляция портальной системы кобальтового аппарата

Моделирования изображений затруднено необходимостью симуляции такого же количества частиц, что и при получении реальных изображений

Симуляция портальной системы аппарата Co-60

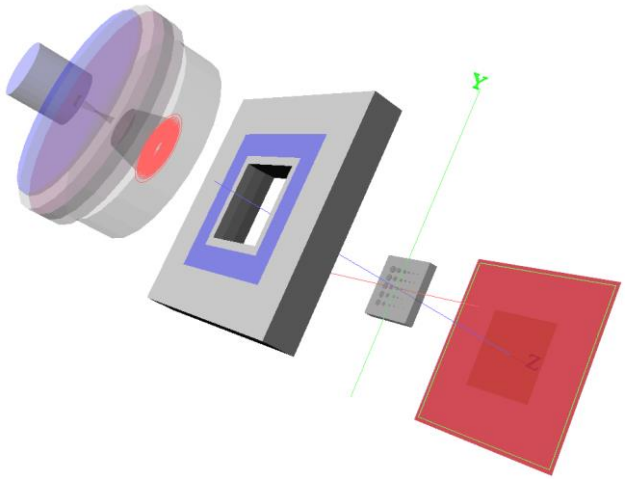


Рисунок 1. Схема аппарата в режиме получения портальных изображений.

- Специальный коллиматор источника позволяет добиваться разрешения, сопоставимого с ускорителями
- Излишнее клиширование приводит к неоправданному повышению шума

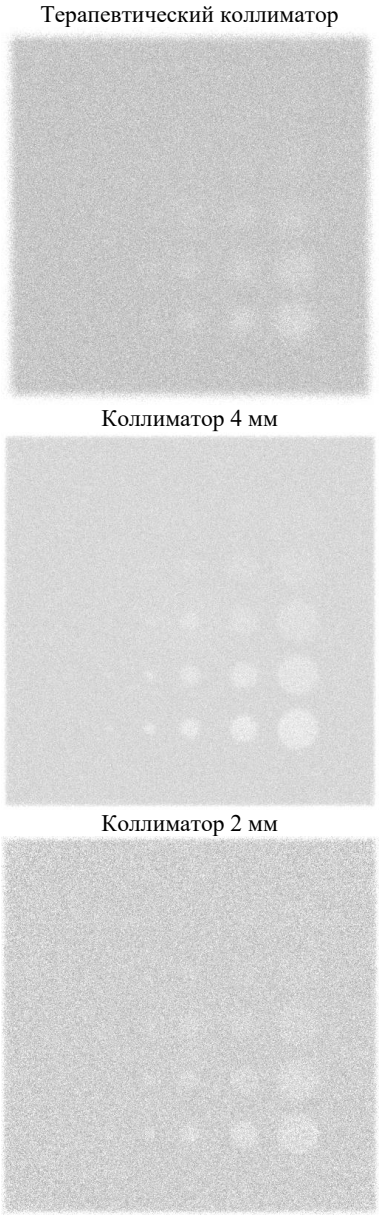
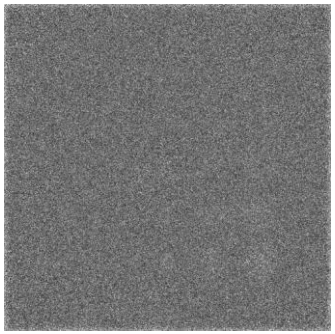
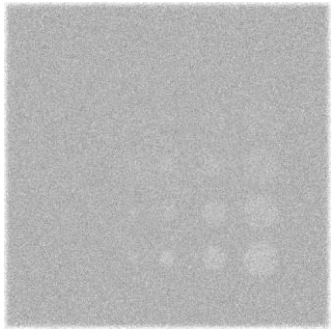


Рисунок 1. Примеры изображений тестового фантома Las Vegas, симулированным с различными коллиматорами. Изображение с минимальным коллиматором симулировано при количестве частиц в 10 раз меньше чем в других случаях.

Симуляция портальной системы аппарата Co-60



10^10



10^11

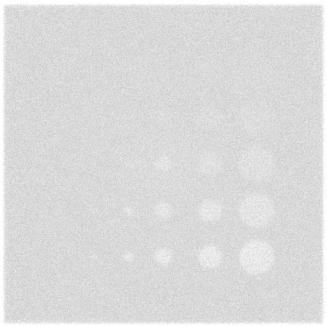
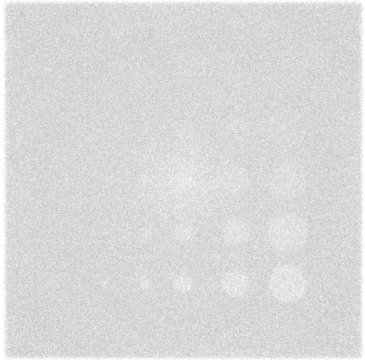


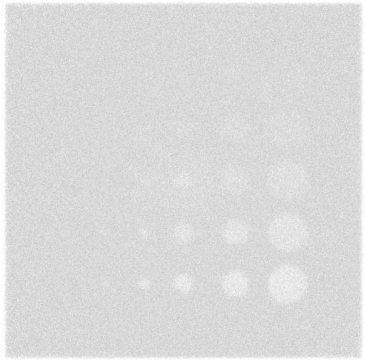
Рисунок 1. **Зависимость качества изображений от количества фотонов. В подписях к изображением указано количество фотонов, извлеченных из модели источника. Для сравнения с реальными условиями за 4 секунды в формировании изображения будут участвовать примерно в 10 раз больше фотонов. В результате качество изображений ожидается по крайней мере сопоставимым по качеству с лучшими поставляемыми с ускорителями системами.**

Изображение без фильтра



Фильтр из водоподобного материала

Al



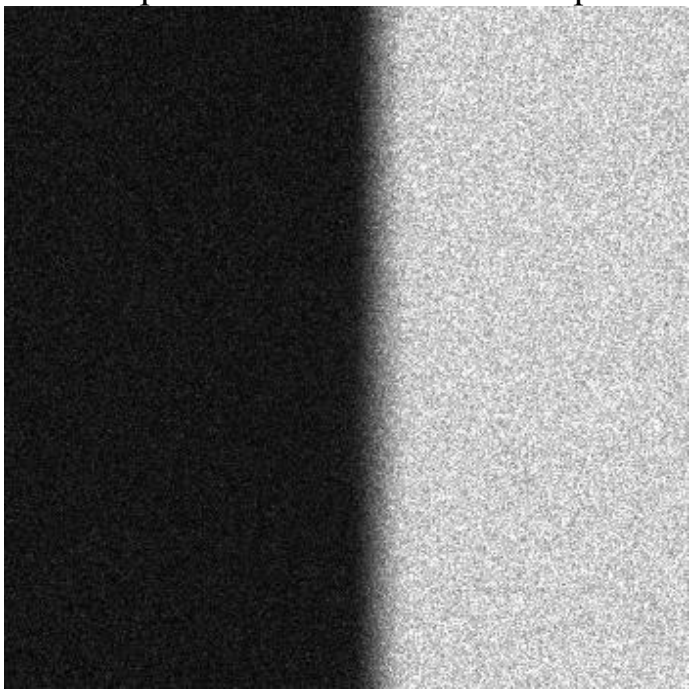
Cu

Pb

Рисунок 1. **Примеры изображений тестового фантома Las Vegas, симулированным при различных материалах фильтров, прилегающих к детектору. Визуально изображения не различимы за**

Симуляция портальной системы аппарата Co-60

Терапевтический коллиматор



Коллиматор 4 мм

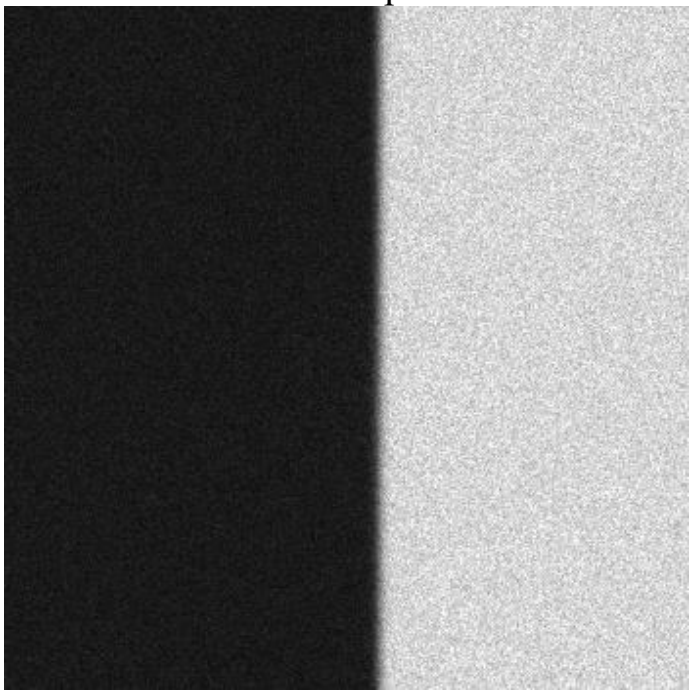


Рисунок 1. Симуляция изображений наполовину закрытых вольфрамовой пластиной толщиной 2 см с целью определения характеристики высококонтрастного разрешения. Слева изображение без специального коллиматора. Справа изображение с портальным коллиматором. Ширина изображений на уровне изоцентра 10 см.

Направления развития

Вопрос точности моделирования процессов взаимодействия ионизирующего излучения с веществом остается открытым.

Транспорт в 3D матрице плотностей.

Тестирование рассеяния электронов

Основные статьи:

1. Ross, C. K.; McEwen, M. R.; McDonald, A. F.; Cojocaru, C. D.; Faddegon, B. A. (2008): Measurement of multiple scattering of 13 and 20 MeV electrons by thin foils. In Med. Phys. 35 (9), p. 4121. DOI: 10.1118/1.2968095兴.
2. Faddegon, Bruce A.; Kawrakow, Iwan; Kubyshin, Yuri; Perl, Joseph; Sempau, Josep; Urban, Laszlo (2009): The accuracy of EGSnrc, Geant4 and PENELOPE Monte Carlo systems for the simulation of electron scatter in external beam radiotherapy. In Phys. Med. Biol. 54 (20), pp. 6151–6163. DOI: 10.1088/0031-9155/54/20/008.
3. Vilches, M.; García-Pareja, S.; Guerrero, R.; Anguiano, M.; Lallena, A. M. (2009): Multiple scattering of 13 and 20 MeV electrons by thin foils: A Monte Carlo study with GEANT, Geant4, and PENELOPE. In Med. Phys. 36 (9), p. 3964. DOI: 10.1118/1.3183501兴.
4. EPAPS Document No. E-MPHYA6-35-034809 for several of the measured scatter distributions from which the results presented in this paper are derived. For more information on EPAPS, see http://www.aip.org/pubservs/epaps.html.

Тестирование рассеяния электронов

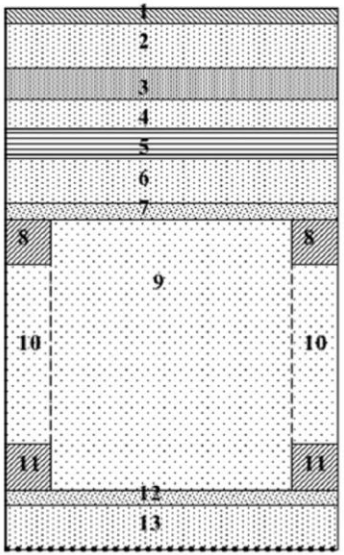


Figure 1. Experimental geometry, as simulated. Drawing is not to scale. Positions of the different components are listed in table 1.

Table 1. Components of the experimental geometry (figure 1). Position is the distance from the front (evacuated side) of the exit window to the front of each component.

Component	Material	Thickness (cm)	Position (cm)	Density (g cm^{-3})	Composition or radius (cm)
1. Exit window	Ti	0.004 12	0	4.42	90% Ti, 6% Al, 4% V
2. Air	Air	2.645 88	0.004 12	0.001 205	75.52% N, 23.18% O, 1.283% Ar, 0.0124% C
3. Scattering foil	See table 2	t	2.65		
4. Air	Air	$2.35 - t$	$2.65 + t$	0.001 205	
5. Monitor chamber	Mylar	0.011 27	5.0	1.40	$H_4C_5O_2$
6. Air	Air	1.486 23	5.011 27	0.001 205	
7. Mylar bag	Mylar	0.0025	6.4975	1.40	
8. Aluminum ring	Al	1.40	6.5	2.70	20.0–23.3 cm
9. Helium between mylar	He	110.0	6.5	0.000 166	0–20 cm
10. Helium between Al	He	107.2	7.9	0.000 166	20–23.3 cm
11. Aluminum ring	Al	1.40	115.1	2.70	20.0–23.3 cm
12. Mylar bag	Mylar	0.0025	116.5	1.40	
13. Air	Air	1.6975	116.5025	0.001 205	
14. Scoring plane	N/A	N/A	118.2	N/A	

Тестирование рассеяния электронов

Наши симуляции:

Сравнение угловых разбросов электронов после различных фольг.

Спасибо!