Московский Государстенный
Университет

Проект по практикуму

Зеркальная комната

Поляков Андрей 341/1

11 мая 2022 г.

Содержание

1 Постановка задачи

Зеркальная комната представляет в плане произвольный замкнутый M-угольник ($4 \leq M \leq 9$), каждая стена – плоское или сферическое зеркало. Для проведения экспериментов необходимо определить для каждой стены комнаты вид зеркала (плоское или сферическое), а для каждого сферического зеркала – его тип (вогнутое или выпуклое) и радиус кривизны.

Основная функция программной системы – проведение оптического эксперимента, при котором из некоторой точки на одной из стен комнаты, под определенным углом к этой стене (угол может варьироваться от 0 до 180 градусов) выпускается луч света, и затем показывается его путь внутри комнаты с учетом отражений от зеркал. Траектория луча определяется физическими законами отражения от зеркальных поверхностей.

Цель моделирования – подбор пользователем системы параметров зеркал и исходного угла выпущенного луча, при которых луч, отражаясь от зеркальных стен, попадает в нужную точку (зону) комнаты.

При визуализации оптического эксперимента должен быть показан план комнаты и изображен путь луча в комнате.

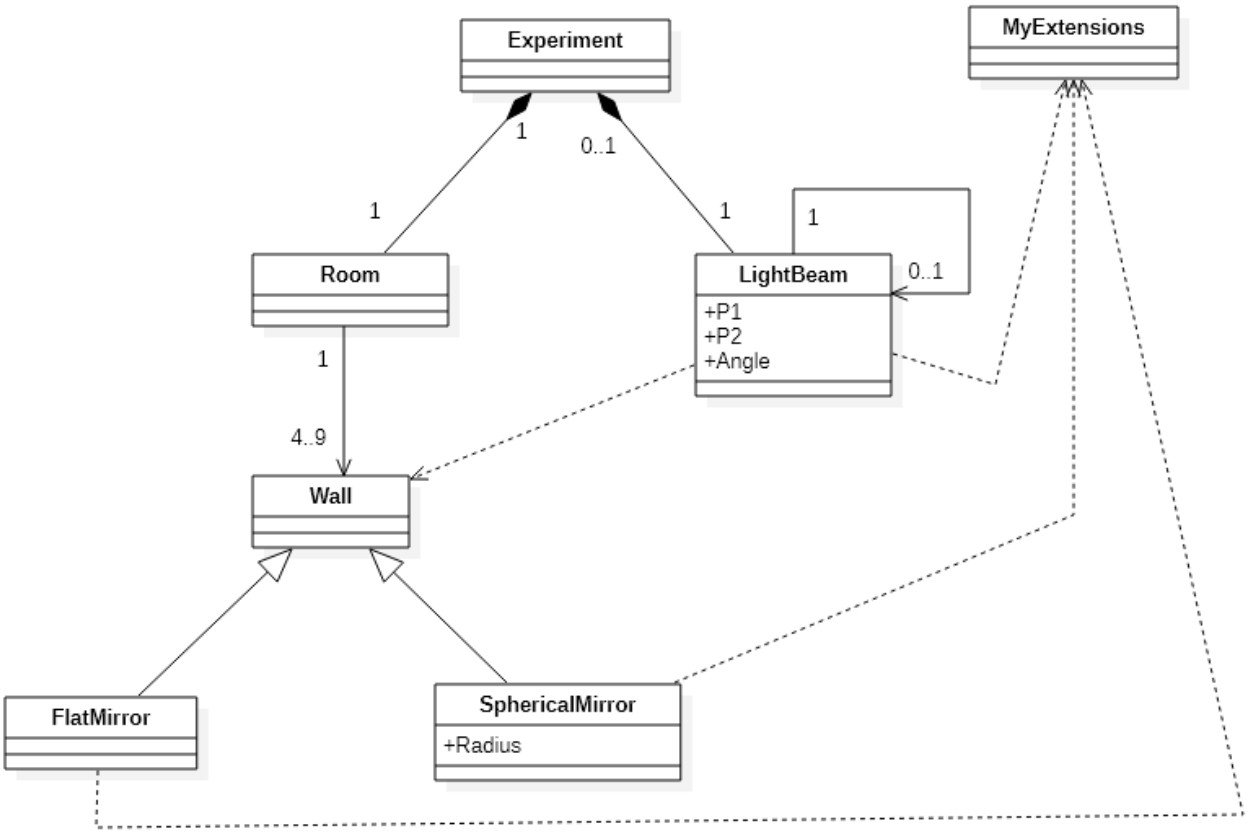
Пользователь системы должен иметь возможность:

- определять число M стен комнаты и рисовать ее план (например, указывая мышью на экране компьютера угловые точки комнаты);
- задавать и изменять параметры зеркал (вид, тип, радиус кривизны), точку выпускания луча и его исходный угол;
- запоминать в файле копию оптического эксперимента, сохраняя все его параметры, и считывать сохраненную копию из файла в рабочее окно.

Требуется, чтобы указанные действия пользователь мог производить в произвольном, удобном для него порядке, и изменение одного параметра эксперимента не должно затрагивать другие установленные параметры.

Возможно усложнение рассматриваемой задачи, когда при отражении от зеркальной поверхности учитывается эффект рассеивания света – в этом случае после нескольких отражений луч становится невидимым. При этом в число параметров эксперимента входят коэффициенты рассеивания каждого зеркала.

2 Диаграмма классов



3 Текстовые спецификации интерфейса

```
public class Experiment
{
    // Форма окна, которому соответствует эксперимент
    public Form1 Form = null;
    // Таймер, по которому эксперимент создает новые лучи (если не мгновенно)
    public System.Timers.Timer timer = new System.Timers.Timer();
    // Генератор случайных чисел
    public Random Random = new Random();
    // Комната эксперимента
    public Room Room;
    // Источник света, задаваемый пользователем
    public LightBeam LightSource = null;
    // Расстояние, в пределах которого засчитывается попадание луча в угол комнаты
    public int PointStopDistance = 2;
    // Мгновенно ли вычисляется эксперимент
    public bool Instant = false;
    // Количество шагов эксперимента (если мгновенные вычисления)
    public int InstantSteps = 10;
    public Experiment(Form1 form, int wallsCount);
    // Инициализация комнаты начальными параметрами
    public void Initialize(int wallsCount, int width, int height);
    // Расчет лучей по таймеру
    private void Timer_Elapsed(object sender, System.Timers.ElapsedEventArgs e);
    // Генерация следующего луча
    private void GenerateNextLightBeam();
}
public class Room
{
    // Эксперимент, соответствующий комнате
    public Experiment Experiment;
    // Список стен комнаты
    public List<Wall> Walls = new List<Wall>();
    public Room(Experiment experiment);
    // Добавление стены
    public void AddWall(Wall wall);
    // Изменение типа стены
    public void ChangeWallType(int index);
}
```

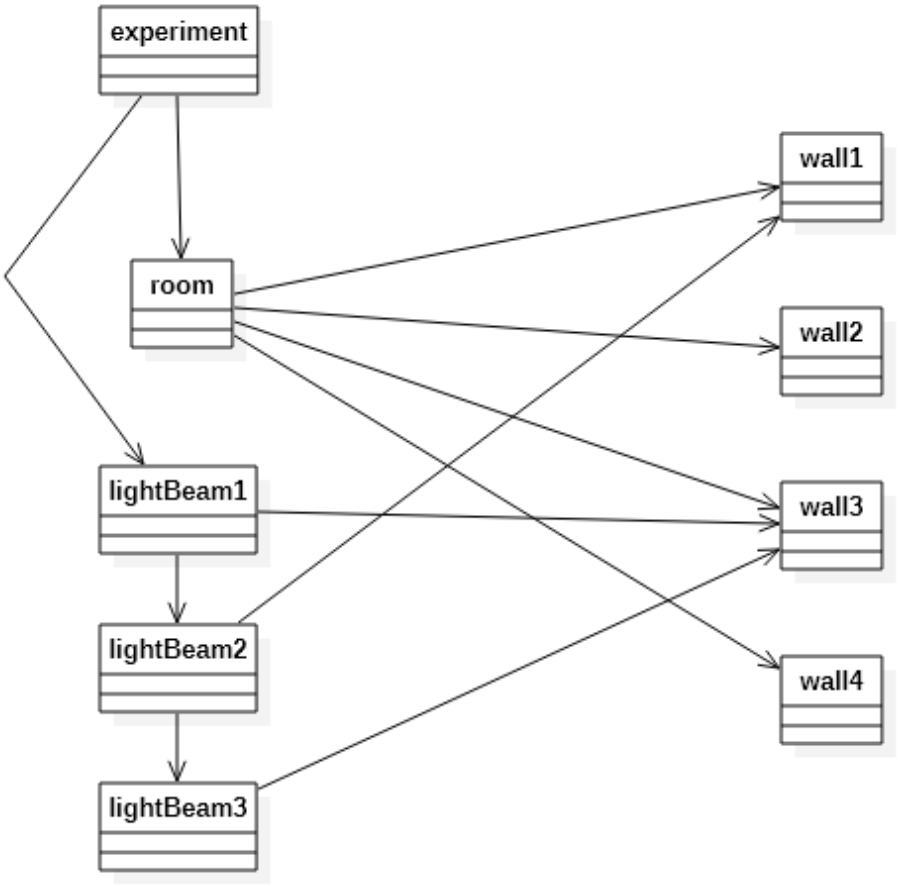
```
public abstract class Wall
{
    // Эксперимент, соответствующий стене
    public Experiment Experiment;
    // Начальная точка стены
    public Point P1 = new Point(0, 0);
    // Конечная точка стены
    public Point P2 = new Point(0, 0);
    // Центр отрезка от начальной точки до конечной
    public PointF SegmentCenter { get; }
    public Wall();
    public Wall(Point p1, Point p2);
    // Отрисовка стены
    public abstract void Draw(Graphics g);
    // Пересечение стены лучём
    public abstract bool Intersect(Point p1, Point p2, List<PointF> intersections);
    // Отражение вектора относительно стены
    public abstract Point Reflect(Point p1, Point p2);
    // Чтение стены из потока данных
    public virtual void Read(BinaryReader br);
    // Запись стены в поток данных
    public virtual void Write(BinaryWriter bw);
    // Чтение стены любого типа из потока данных
    public static Wall Read(Experiment experiment, BinaryReader br);
}
public class FlatMirror : Wall
{
    public FlatMirror();
    public FlatMirror(Point p1, Point p2);
    // Отрисовка стены
    public override void Draw(Graphics g);
    // Пересечение стены лучём
    public override bool Intersect(Point p1, Point p2, List<PointF> intersections);
    // Отражение вектора относительно стены
    public override Point Reflect(Point p1, Point p2);
    // Запись стены в поток данных
    public override void Write(BinaryWriter bw);
}
```

```
public class SphericalMirror : Wall
{
    // Радиус кривизны
    public float Radius { get; set; }
    // Точка середины дуги
    public Point P3;
    // Радиус зоны взаимодействия с центром дуги
    public const int P3delta = 10;
    // Точка центра окружности
    public PointF CircleCenter;
    // Начальное значение угла в радианах
    public float AngleBegin;
    // Конечное значение угла в радианах
    public float AngleDiff;
    public SphericalMirror();
    public SphericalMirror(Point p1, Point p2, int radius = 1000);
    // Отрисовка стены
    public override void Draw(Graphics g);
    // Пересечение стены лучём
    public override bool Intersect(Point p1, Point p2, List<PointF> intersections);
    // Отражение вектора относительно стены
    public override Point Reflect(Point p1, Point p2);
    // Чтение стены из потока данных
    public override void Read(BinaryReader br);
    // Запись стены в поток данных
    public override void Write(BinaryWriter bw);
}
```

```
public class LightBeam
{
    // Начальная точка сегмента луча
    public Point P1;
    // Конечная точка сегмента луча
    public Point P2;
    // Стена, от которой сегмент луча оттолкнулся
    public Wall CurrentWall = null;
    // Стена, в который сегмент луча уперся
    public Wall Colliding = null;
    // Угол сегмента луча в радианах
    public double Angle;
    // Ссылка на следующий сегмент луча
    public LightBeam Next = null;
    public LightBeam(Point p, double angle);
    public LightBeam(Point p1, Point p2);
    // Рассчитать конечную точку сегмента луча
    public void CalculateEnd(Experiment experiment);
    // Сгенерировать следующий луч
    public LightBeam GenerateNext(Experiment experiment);
}
```

```
// Тип пересечения отрезка
public enum seg_cross_t { seg_crossing = 1, seg_collide, seg_nocross };
// Тип ориентации относительно отрезка
public enum turn_t { left = 1, right = -1, collinear = 0 };
// Тип положения точки относительно контура
public enum point_loc_t { inside, outside };
public static class MyExtensions
{
    // Установка двойной буферизации отрисовки
    public static void SetDoubleBuffered(this Panel panel);
    // Проекция точки на прямую
    public static bool PointToSegmentProject(Point line1, Point line2,
        Point toProject, ref Point project);
    // Расстояние между точками
    public static double PointDistance(Point p1, Point p2);
    public static double PointDistance(PointF p1, PointF p2);
    // Положение точки относительно прямой
    public static int LineSide(Point line1, Point line2, Point p);
    // Проверка на вложенность точки в прямоугольник
    public static bool PointInRect(Point p, Point p1, Point p2);
    // Константы вычислительной погрешности
    const double eps_precision_compare_double = 1e-09;
    public static double machine_epsilon = 1.0d;
    // Сравнение двух вещественных
    public static bool _equal(double x, double y);
    public static bool _less(double x, double y);
    public static bool _more(double x, double y);
    // Вычисление положения точки относительно отрезка
    public static turn_t turn(PointF a, PointF b, PointF c);
    // Вычисление вспомогательной дельты
    public static (float X, float Y, float Z) calc_delta(PointF a, PointF b,
        PointF c, PointF d);
    // Вычисление детерминанта
    public static PointF det(PointF a, PointF b);
    // Вычисление формулы уравнения через две точки
    public static (float X, float Y, float Z) equation(PointF a, PointF b);
    // Ранг матрицы
    public static int rank((float X, float Y, float Z) a,
        (float X, float Y, float Z) b);
    // Проверка вложенности точки в прямоугольник
    public static bool point_in_rect(PointF rect1, PointF rect2, PointF a);
    // Пересечение отрезков
    public static seg_cross_t seg_cross(PointF a, PointF b, PointF c,
        PointF d, ref PointF result);
}
```

4 Диаграмма объектов



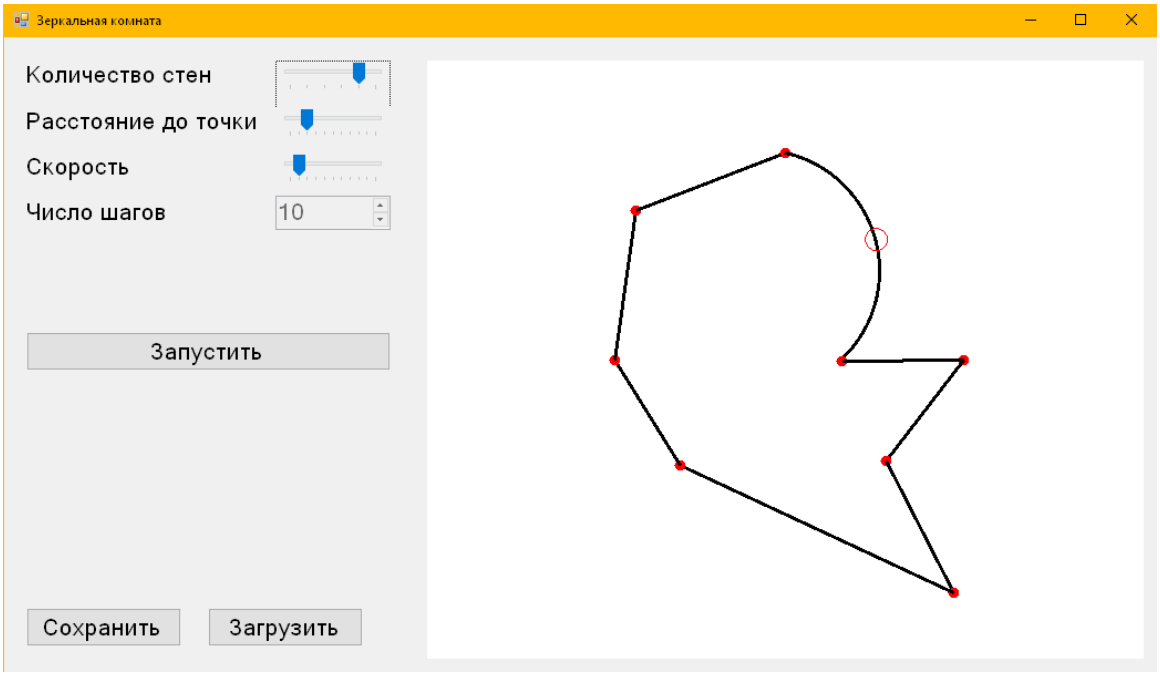
5 Инструментальные стредства

1. Язык разработки - C
2. Среда разработки - Microsoft Visual Studio .NET Framework 4.7.2
3. Используемые библиотеки:
    - System.Windows.Forms

6 Описание файловой структуры системы

1. Program.cs - главный файл, запуск формы
2. Experiment.cs - основной класс эксперимента
3. Room.cs - класс моделируемой комнаты
4. Wall.cs - абстрактный класс стены
5. FlatMirror.cs - класс зеркальной стены, наследующийся от Wall
6. SphericalMirror.cs - класс сферической стены, наследующийся от Wall
7. LightBeam.cs - класс одного отрезка луча света
8. Form1.cs - форма основного окна моделирования
9. MyExtensions.cs - файл вспомогательных расширений и методов

7 Пользовательский интерфейс



В правой части окна находится рабочее поле моделирования. Углы комнаты помечены красными точками, их можно хватать и перетаскивать левой кнопкой мыши. Стены изображены черными отрезками.

Нажатие центральной кнопки мыши на плоскую стену превращает её в сферическую. Нажатие центральной кнопки мыши на красный кружочек в центре дуги сферической стены превращает её в плоскую. Радиус кривизны можно изменять с помощью захвата и перетаскивания левой кнопки мыши красного кружочка в центре дуги сферической стены.

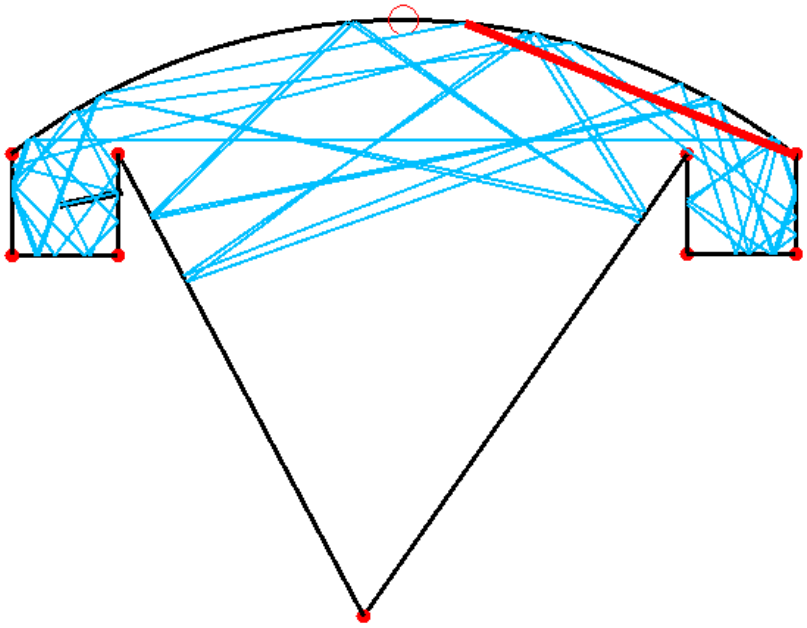
Луч света можно задать, зажав и отпустив правую кнопку мыши в любой точке рабочего поля. Луч будет идти от точки начала нажатия в сторону точки окончания нажатия.

В левой части окна сверху есть несколько полей параметров моделирования:

- Ползунок количества стен (от 4 до 9)
- Ползунок расстояния попадания в угол комнаты
- Ползунок скорости моделирования
- Поле ввода количества шагов моделирования (активно, если скорость $= 0$)

В нижнем левом углу есть кнопки "Сохранить"и "Загрузить позволяющие сохранить техущую конфигурацию комнаты и загрузить соответственно.

8 Описание проведенных экспериментов



В ходе экспериментов было выявлено:

- Эксперимент Роджера Пенроуза повторить не получилось: предполагаемая автором кривизна стены является эллиптической, в то время как данный проект предполагает лишь дугу окружности. Тем не менее, на изображении можно видеть, что лучи света действительно в основном находятся в рамках предполагаемой области
- Ситуация попадания луча в угол комнаты является проблемным местом моделирования, не соответствующим никакой реальной физической ситуации. В рамках данного моделирования была выбрана стратегия остановки моделирования в случае попадания в угол.
- Артефакты моделирования могут происходить при выборе любого сколь угодно малого значения проверки расстояния от текущей точки пересечения до новой точки пересечения. В случае необходимости гарантированного отсутствия артефактов необходима более точная модель.