МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ

ПРИМЕРЫ ПОСТРОЕНИЯ МОДЕЛЕЙ

ἀγεωμέτρητος μηδείς εἰσιτω

Рассмотрим два конкретных примера математических моделей и их программных реализаций.

Пример 1. Прыгающий мяч. Задача

Построить математическую модель движения прыгающего мяча. Мяч падает с высоты y_0 с начальной скоростью v_0. Задан коэффициент восстановления скорости k, равный отношению $|v_2|/|v_1|$, где v_1 — скорость мяча в момент времени, непосредственно предшествующий удару о землю, v_2 — скорость сразу после удара (предполагается, что скорость при ударе меняется мгновенно), $0 \leqslant k < 1$. (Крайние значения коэффициента восстановления соответствуют абсолютно упругому *($k = 1$) и* абсолютно неупругому *($k = 0$) ударам.)*

Рассмотреть варианты, когда не учитывается и когда учитывается сопротивление воздуха. В последнем случае считать, что сопротивление среды пропорционально скорости мяча, коэффициент сопротивления $— u, u > 0$. Поверхность падения считать идеально гладкой и строго горизонтальной.

Сделать графическую анимированную иллюстрацию модели. Для первого случая (когда сопротивление воздуха не учитывается) вывести график зависимости координаты от времени, для второго — зависимость силы сопротивления от времени. Входные параметры модели: y_0, v_0, k, u.

Формулировка задачи уже содержит словесное описание модели, поэтому, перейдём ко второму этапу построения модели — идеализации. Перечислим упрощения, которые будут присутствовать в модели. Поскольку поверхность удара строго горизонтальная и идеально гладкая, мяч будет двигаться вертикально, не отклоняясь в стороны после ударов о землю, и без кручения. Поэтому можно считать мяч материальной точкой, масса мяча сосредоточена в этой точке — его «центре». (В случае, когда принимается во внимание сопротивление воздуха, модель, очевидно, должна учитывать площадь поперечного сечения мяча, но удобно считать, что эта величина входит в качестве множителя в коэффициент u.) В соответствии с законом всемирного тяготения на мяч действует сила

$$wm = \gamma \frac{mM}{(R+y)^2}, \tag{1}$$

где M, m — массы соответственно Земли и мяча, R — радиус Земли, y — высота подъёма мяча над поверхностью Земли, w — ускорение мяча, γ — константа. Поскольку, очевидно, в данном случае, что y много меньше R, то отсюда следует, что можно принять

$$w = \gamma \frac{M}{R^2} = \text{const} \stackrel{\text{def}}{=} g. \tag{2}$$

Константу g называют *ускорением свободного падения*.

На третьем и четвёртых этапах, пользуясь вторым законом Ньютона, спроектируем силы, действующие на мяч, на вертикальную ось, направленную вверх, с началом в точке удара. Тогда при отсутствии сопротивления воздуха получим

$$\ddot{y} = -g, \qquad y(t_0) = y_0, \quad \dot{y}(t_0) = v_0. \tag{3}$$

Решение этого дифференциального уравнения

$$y = -\frac{1}{2}\,gt^2 + C_1 t + C_2, \tag{4}$$
$$C_1 = gt_0 + v_0, \quad C_2 = y_0 + \frac{1}{2}\,gt_0^2 - C_1 t_0$$

полностью определяет координату мяча y как функцию времени. Скорость мяча

$$\dot{y} = -gt + C_1. \tag{5}$$

Таким образом, модель описывается функцией (4), меняющуюся на временных отрезках $[0, t_1], [t_1, t_2], \ldots$, где t_i — моменты удара мяча о землю (начальный момент времени всегда можно взять равным нулю), причём после i-го удара начальные условия будут такими: $y_{0i} = 0$, $v_{0i} = -kv_i$, где v_i — скорость непосредственно перед i-ым ударом. Легко видеть, что в построенной модели, при $k > 0$, мяч никогда не остановится, поэтому задаваемую величину количества ударов следует присоединить к входным параметрам модели.

Для случая, когда учитывается сопротивление среды аналогично получаем уравнение

$$\ddot{y} = -g - u\dot{y}, \qquad y(t_0) = y_0, \quad \dot{y}(t_0) = v_0, \tag{6}$$

имеющее решение

$$y = -\frac{D_1}{u}\, e^{-ut} - \frac{g}{u}\, t + D_2, \tag{7}$$

$$D_1 = \left(v_0 + \frac{g}{u}\right) e^{ut_0}, \quad D_2 = y_0 + \frac{D_1}{u}\, e^{-ut_0} + \frac{g}{u}\, t_0.$$

В этом случае скорость мяча определяется соотношением

$$\dot{y} = D_1\, e^{-ut} - \frac{g}{u}, \tag{8}$$

и, кроме того, необходимо выполнение неравенства

$$uv_0 > -g, \tag{9}$$

иначе сила сопротивления воздуха будет уравновешивать или превосходить силу притяжения.

✎ *Замечание*. В данном простом учебном примере получены аналитические решения дифференциальных уравнений, описывающие рассматриваемую модель. Часто в наиболее интересных для исследования случаях аналитические решения получить не удаётся и, поэтому, приходится прибегать к *численным решениям*. Следовательно, при этом дополнительно встаёт задача выбора и обоснования подходящих численных методов. Вычислительный алгоритм таких решений должен удовлетворять жёстким, иногда противоречивым, требованиям, как-то: необходимость получения решения за разумное (по возможности — минимальное) количество действий и время при заданной точности; объёмы обрабатываемой информации не должны превышать ёмкости машинной памяти; структура алгоритма должна быть достаточно простой при учёте архитектуры вычислительной системы, вычислительный процесс должен быть устойчивым (ошибки округления не должны накапливаться в процессе счёта) и т. д. Причём данные требования далеко не всегда удовлетворяются автоматически по мере роста мощностей компьютеров из-за постоянно возрастающей сложности решаемых задач. По этим причинам разработка эффективных вычислительных алгоритмов является одной из ключевых задач математического моделирования. По теории численных методов рекомендуются книги [4], [5], [6].

Копьютерная реализация построенной математической модели.

1. Мы должны иметь два объекта, которые назовём Y1 (без учёта сопротивления воздуха) и Y2 (с учётом сопротивления). В нашем случае легко видеть, что оба объекта имеют сходное поведение. Удобно вести расчёты отдельно для каждого из временных промежутков $[0, t_1]$, $[t_1, t_2]$, …, $[t_i, t_{i+1}]$, …, где t_i — моменты удара мяча о землю. Поскольку в условии задачи сказано об анимационной иллюстрации движения мяча, необходимо знать: координату мяча в каждый заданный момент времени, время удара о землю, скорость при ударе, максимальное значение координаты (неоходима для масштабирования окна графического вывода). Методы (обработчики сообщений), возвращающие эти параметры, очевидно, будут общими для обоих объектов. Для объекта Y2, сверх того, нужно предусмотреть возврат значения скорости в каждый момент времени.

2. Спроектируем соответствующую иерархию классов. В базовом классе Tcoordinate сосредоточим общие обработчики сообщений, возвращающие координату мяча в каждый заданный момент времени get_top_list, а также время удара get_t_impact, скорость при ударе get_v_impact, максимальную координату get_Max_value. Перечисленных методов, очевидно, достаточно и для функционирования класса Ty1, а для класса Ty2, кроме них, потребуется возвращать значение скорости (get_v_list).

3. Количество точек N для вычислений на каждом из временных интервалов $[t_i, t_{i+1}]$ разумно предоставить задавать пользователю, присоединив эту величину к множеству входных параметров модели. Для того, чтобы время, затрачиваемое на расчёты координат, не влияло на время анимационного цикла на временных промежутках между ударами о землю, удобно все вычисления сделать до начала этого цикла, а при анимации считывать уже готовые значения. Следовательно, вычисленные значения координат нужно хранить в списке, выделяя и возвращая под него память динамически на каждом временном интервале — метод calc_values, а метод get_top_list будет возвращать указатель на «вершину» этого списка top_coord_list. Для определения момента удара t_impact и скорости в момент удара v_impact, а также максимальной координаты max_coord_value, потребуются соответствующие методы calc_t_impact, calc_v_impact и calc_max_value. Для вычислений координаты и скорости нужно предусмотреть соответствующие методы — position, velocity, причём в базовом классе они должны быть абстрактными и виртуальными, так как соответствующяя информация будет доступна только в призводных классах.

В классе Ty2 потребуется метод calc_v, рассчитывающий скорости точки в заданные моменты времени, и сохраняющий их в списке, а метод get_v_list будет возвращать указатель на «вершину» этого списка top_v_list.

Поддержка анимации и графики вынесена в отдельный модуль, так как эти средства являются внешними по отношению к рассматриваемым объектам, системно зависимыми, и могут быть реализованы различными способами, например, средствами библиотек OpenGL или DirectX.

Объявления классов. Базовый класс

```
type Tcoordinate = class
protected
t0                  : double;          // начальные значения времени
y0                  : double;          // координаты;
v0                  : double;          // и скорости;
k                   : double;          // коэффициент восстановления;
t_impact            : double;          // момент удара;
v_impact            : double;          // скорость в момент удара;
max_coord_value : double;              // макс. значение координаты;
top_coord_list      : value_list ;     // указатель на "верх" списка;
N                   : LongInt;         // количество точек расчёта;
function    position( t: double ): double; virtual; abstract;
function    velocity( t: double ): double; virtual; abstract;
procedure   calc_values;               // опр. значений координат;
procedure   calc_t_impact;             // опр. момента удара о "пол";
procedure   calc_v_impact;             // опр. скорости при ударе;
procedure   calc_max_value;            // опр. максимальной высоты;
public
function    get_top_list      : value_list ;
function    get_t_impact      : double;
function    get_v_impact      : double;
function    get_Max_value : double;
```

```
constructor  birth( tt0, yy0, vv0, kk: double; NN: LongInt );
destructor   kill; virtual;
end;
```

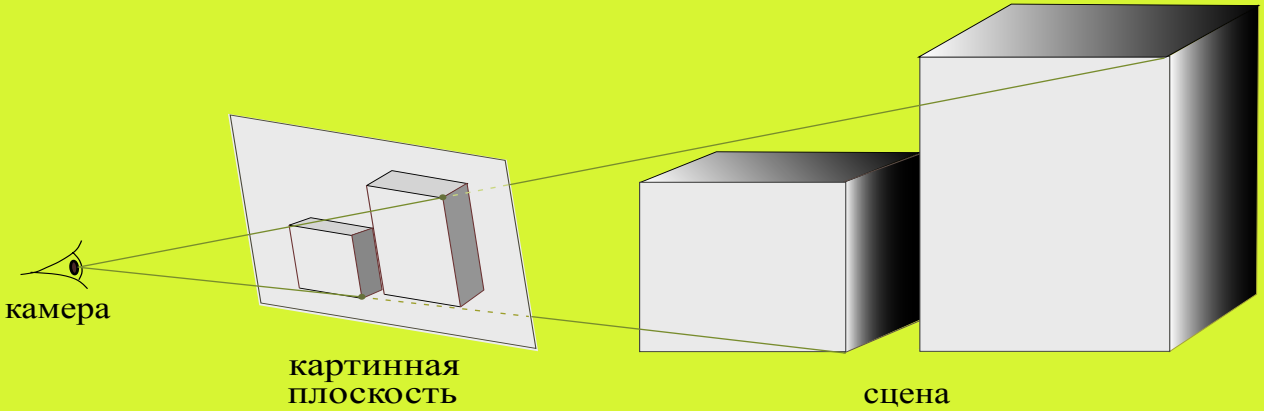
Классы-наследники

```
type Ty1 = class( Tcoordinate )
protected
C1 : double;                                          // константы
C2 : double;                                          // интегрирования;
function     position( t: double ): double; override;
function     velocity( t: double ): double; override;
public
constructor  birth( tt0, yy0, vv0, kk: double; NN: LongWord );
destructor   Kill; override;
end;
```

```
type Ty2 = class( Tcoordinate )
 protected
 C1          : double;              // константы
 C2          : double;              // интегрирования;
 u           : double;              // коэфф. сопротивления среды;
 top_v_list  : value_list ;         // указатель на "верх" списка;
 procedure   calc_v;                // определение скоростей точки;
 function    position( t: double ): double; override;
 function    velocity( t: double ): double; override;
 public
 function    get_v_list : value_list ;
 constructor birth(tt0, yy0, vv0, kk, uu: double;NN: LongWord);
 destructor  Kill; override;
 end;
```

Программа в действии.

Пример 2. Реалистичное освещение. Рассмотрим модель реалистичного освещения трёхмерной сцены, основанную на *трассировке* лучей света. Прежде чем сформулировать задачу приведём все необходимые определения, относящиеся к трёхмерной компьютерной графике и трассировке лучей. Поскольку русская терминология в этой области в настоящее время недостаточно установилась, все названия ниже будут сопровождаться своими английскими аналогами. Все объ-



екты в совокупности, которые будут отрисованы программой называются частями *сцены* или *мира*. В компьютерной графике точка, из которой осуществляется наблюдение называется, *камерой* (camera). По аналогии с фотокамерой, на плёнку которой проецируется и записывается сцена, в графике мы имеем *картинную плоскость* (view window), на которую проецируется трёхмерная сцена и затем отрисовывается.

Каждый пиксел (pixel — точка растрового изображения) полученной картинки является следствием программной имитации светового луча, который попадает на картинную плоскость по пути в камеру. Задача получения изображения сцены заключается, таким образом, в определении цвета каждой точки картинной плоскости.

Для решения этой задачи в компьютерной графике применяются самые различные методы. Одним из таких методов, потенциально позволяющий добиться максимально возможного *реализма изображения*, является метод *трассировки лучей* света — *рейтрессинг* (ray tracing). Рейтрессинг называется так (tracing — прослеживание, запись регистрирующего прибора), потому что при использовании этого метода отслеживается путь, которым лучи света распостраняются на сцене — *трассируют* сцену. Целью трассировки является определение цвета каждого луча, который попадает на картинную плоскость после всех отражений и преломлений от объектов сцены, перед тем тем как он достигнет камеры.

Если отслеживать все лучи с исходной точки источника света (light source) на всём протяжении пути до камеры, то такой наиболее точный способ окажется слишком трудоёмким из-за полномасштабных численных расчётов. К тому же, при этом многие лучи, исходящие от осветителя, очевидно, вообще не попадут в камеру. Поэтому вместо трассировки от источника света, обычно применяют метод *обратной трассировки* — лучи трассируют в обратном порядке, начиная от камеры. Таким образом, метод обратной трассировки делает то же самое, что и оригинальный способ, за исключением того, что понапрасну не тратятся усилия на лучи, которые никогда не достигают камеры.

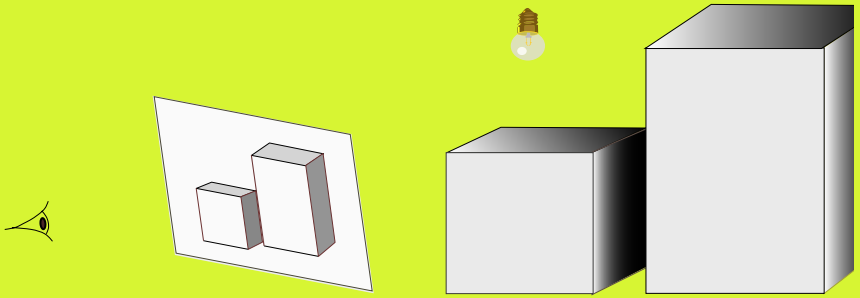


Рис. 1. Прямая трассировка

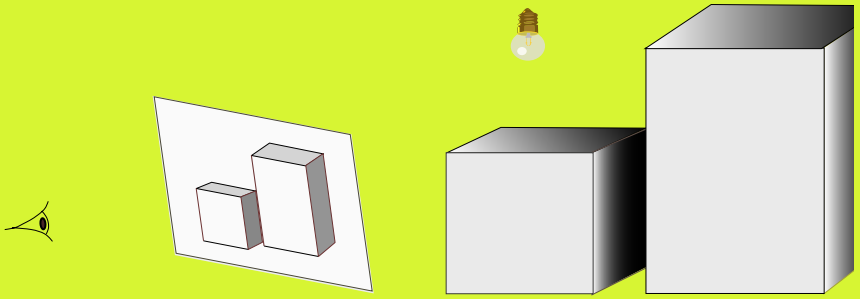


Рис. 2. Обратная трассировка

Определение цвета объекта в точке пересечения луча с его поверхностью зависит от выбранной *модели освещённости*. Опишем одну из самых распространённых — *модель Фонга* (Phong). В этой модели общая интенсивность (которая определяется *энергией* световой волны, обычно принимается, что величина интенсивности меняется от 0 до 1) освещения складывается из следующих трёх компонент:

- интенсивности *рассеянного* (ambient) освещения;
- интенсивности *диффузного* (diffuse) освещения;
- интенсивности *зеркального* (specular) освещения.

Интенсивность рассеянного (фонового) освещения I_a обусловлена множественными отражениями света от всех объектов сцены, она считается постоянной в любой точке пространства

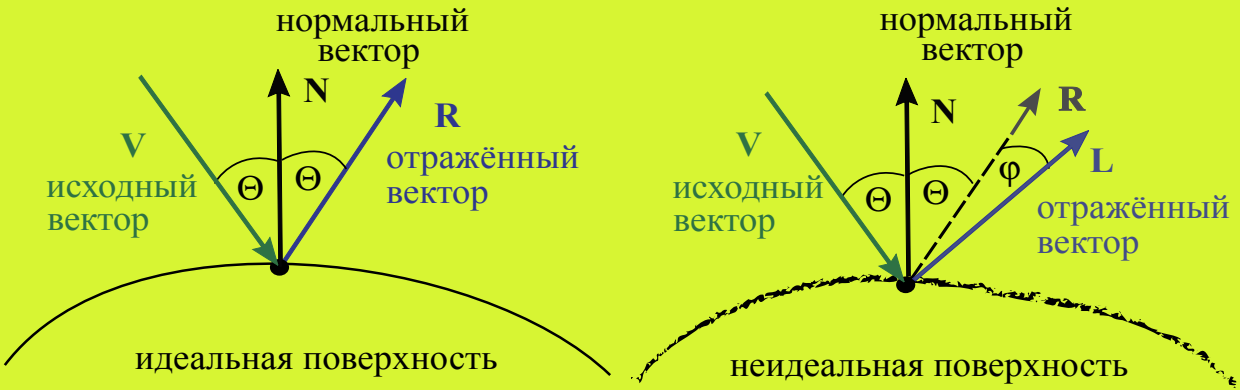
$$I_a = k_a I_p,$$

где I_p — интенсивность источника света, коэффициент $k_a \in [0; 1]$.

Интенсивность диффузного освещения I_d можно рассчитать на основе *закона Ламберта*:

$$I_d = k_d I_p \cos \Theta,$$

где Θ — угол падения луча на поверхность, k_d — коэффициент диффузного отражения, $k_d \in [0; 1]$, определяет меру «шершавости» или «зернистости» поверхности объекта.



Интенсивность I_s зеркально отражённого от поверхности света вычисляется в зависимости от степени отклонения от направления отражённого луча в идеальном случае (зеркало).

$$I_s = k_s I_p \cos^n \varphi,$$

где k_s — коэффициент зеркальности $k_s \in [0; 1]$, n определяет степень «зеркальности» поверхности (для зеркала $n \to \infty$).

Общая интенсивность света I по Фонгу в каждой точке при наличии m источников света определяется соотношением

$$I = I_a + \sum_{k=1}^{m} \left(I_{d,k} + I_{s,k} \right).$$

Итак, *метод трассировки лучей состоит в следующем*: для каждого пиксела на картинной плоскости, определяется луч, проходящий от камеры к этой точке. Этот луч прослеживается по всей сцене при всех его отражениях от различных объектов и преломлениях в соответствии с законами физики. Окончательный цвет луча (и, следовательно, соответствующего пиксела картинной плоскости) определяется цветами объектов, на которые падает луч при прохождении через сцену, и цветом источников света.

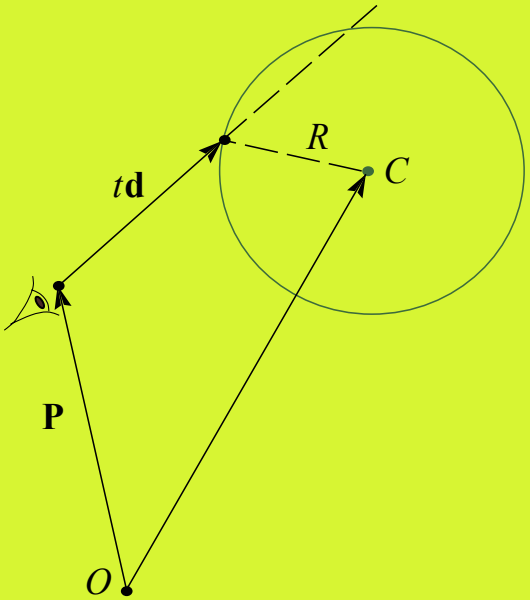
Теперь сформулируем задачу

Построить математическую и компьютерную модели освещения пространственной трёхмерной сцены на основе метода трассировки лучей света. Цвет каждой точки объектов сцены определять на основе модели Фонга. Ограничиться только сферами и одним источником света. Сферы считать полностью непрозрачными. Ослабление интенсивности источника света с расстоянием не учитывать. Картинную плоскость считать фиксированной в пространстве. Входные параметры модели: количество сфер, их радиусы, цвета, коэффициенты диффузии и отражения, координаты источника света, его цвет, координаты положения камеры.

Формулировка задачи уже содержит словесное описание модели, поэтому, перейдём ко второму этапу построения модели — идеализации. Перечислим упрощения, которые будут присутствовать в модели. Поскольку все объекты сцены полностью непрозрачны, то преломление света в данной модели не учитывается, учитывается только отражение.

Закон отражения света: отражённый луч лежит в плоскости падения и угол отражения равен углу падения, или, обозначая $\mathbf{v}, \mathbf{n}, \mathbf{r}$ направления соответственно падающего на поверхность луча, нормали к поверхности и отражённого луча (все векторы нормированы), получим соотношение

$$\mathbf{r} = \mathbf{v} - 2(\mathbf{n}, \mathbf{v})\mathbf{r}, \qquad (10)$$



справедливость которого легко проверить подставив $\mathbf{r}$ в формулу, выражающую закон отражения: $(\mathbf{n}, \mathbf{r}) = -(\mathbf{n}, \mathbf{v})$. Для того, чтобы определить факт пересечения луча со сферой, обозначим $\mathbf{d}$ направление луча из камеры на точку пересечения, $|\mathbf{d}| = 1$. Тогда из уравнения

$$\left| -\overrightarrow{OC} + \mathbf{P} + t\mathbf{d} \right| = R,$$

обозначив $\mathbf{V} = \mathbf{P} - \overrightarrow{OC}$, получим

$$(\mathbf{V} + t\mathbf{d}, \mathbf{V} + t\mathbf{d}) = R^2,$$

откуда находим

$$t_{1,2} = \frac{-(\mathbf{V},\mathbf{d}) \pm \sqrt{(\mathbf{V},\mathbf{d})^2 - \mathbf{d}^2(\mathbf{V}^2 - R^2)}}{\mathbf{d}^2}. \qquad (11)$$

Отсюда следует, что при отрицательном значении величины, стоящей в (11) под знаком радикала, луч не пересекает сферу, в противном случае, беря меньший *положительный* корень, получим радиус-вектор точки пересечения: $\mathbf{P} + t\mathbf{d}$.

Таким образом, *алгоритм модели* таков. Луч «выстреливается» в заданном направлении, чтобы определить имеется ли там что-нибудь. По формуле (11) определяются точки пересечения луча со всеми сферами сцены. После того, как получим все точки пересечения, выбираем из них ближайшую и вычисляем освещённость объекта в данной точке. Для этого нужно:

- выполнить тест затенения, определяющий освещает источник света точку пересечения или нет;
- найти вектор нормали к поверхности объекта в точке пересечения. Он определяет диффузный компонент освещения, а также направление отражённого луча;
- найти отражённый луч, который определяет зеркальный компонент освещения и, конечно, цвет отражённого света (если объект отражает свет).

После чего цвет точки определяется на основе модели Фонга. После отражения луча эта процедура повторяется и т. д.

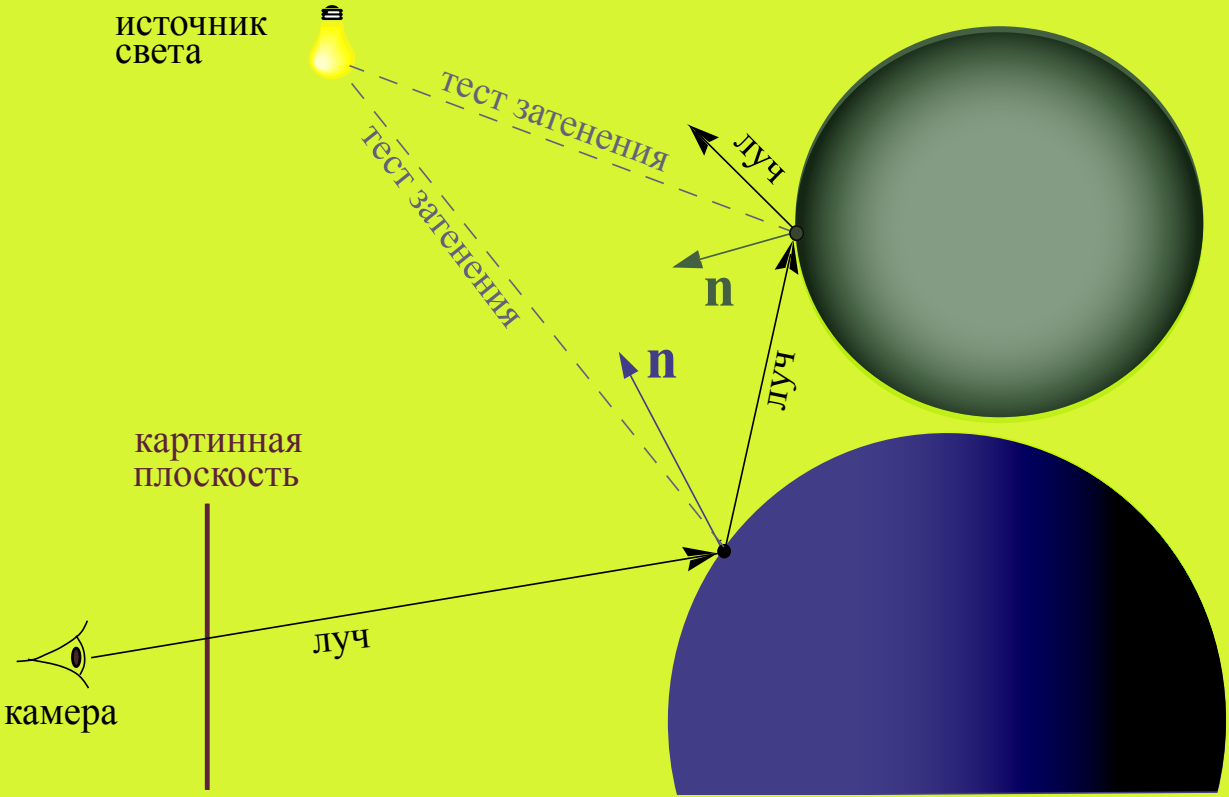


Рис. 3. Алгоритм трассировки

Рассмотрим программную реализацию построенной нами математической модели, не останавливаясь подробно на вопросах проектирования иерархии классов, т. к. её вид в данном случае вполне очевиден.

Для того, чтобы иметь возможность работать с лучами и задавать координаты точек в пространстве нужны *векторы* — соответствующие процедуры и функции сосредоточим в классе Tvector. Для работы с *цветами* нам понадобятся соответствующие процедуры и функции смешивания цветов — модуль `colorUnit`. Объект Луч, который нужен для работы с источником света, по существу является вектором, имеющим цвет, поэтому выведем соответствующий класс Tray из класса Tvector.

```
Type Tray = class( Tvector )
 protected
 color                    : Tcvet;
 public
 function      get_source : Tvector;
 function      get_color  : Tcvet;
 constructor   birth( x0, y0, z0: double;
                      RR, GG, BB: integer ); overload;
 constructor   birth( p0 :        Tpoint3d;
                      CC :        Tcvet     ); overload;
 end;
```

Несмотря на то, что в нашей модели только один тип тел — сфера, учитывая возможные дальнейшие модификации программы, когда к сферам на сцену могут добавиться и другие виды объектов, сосредоточим общие для них методы в базовом классе Tfigure. Этот класс содержит методы для расчёта отражённого луча, вычисления коэффициентов диффузии и отражения в заданной точке, для которых нужно знать *нормаль* к данной точке. Поскольку нормаль можно определить только для конкретного тела, то метод set_normal объявлен виртуальным и абстрактным, с тем, чтобы в дальнейшем иметь возможность его переопределения в производных классах.

```
type  Tfigure = class
protected
center      : Tvector;          // центр фигуры;
normal      : Tvector;          // нормаль;
reflected   : Tvector;          // отражённый луч;
color       : Tcvet;            // цвет;
n_spec      : double;           // показатель отражения;
v_spec      : double;           // коэффициент отражения;
procedure set_normal( point: Tvector ); virtual; abstract;
procedure set_reflected( incident, p: Tvector );
public
function  get_reflected( incident, p: Tvector ): Tvector;
function  get_diff( p, light_direction : Tvector ): double;
function  get_spec( p, ray_direction: Tvector;
                    light_direction : Tvector ): double;
```

```
function  get_value_spec: double;
function  get_color: Tcvet;
constructor  birth( center0: Tvector; color0: Tcvet;
                    n_spec0, v_spec0: double          );
end;
```

Из класса Tfigure выведем класс Tsphere (сфера). Объект этого класса должен иметь возможность рассчитывать вектор нормали к любой точке своей поверхности, определять факт пересечения трассирующего луча с поверхностью и, кроме того, определять расстояние до камеры. Для этого в классе Tsphere заданы методы set_normal и intersection, последний из которых возвращает множитель t, на который нужно умножить направляющий вектор трассирующего луча, чтобы этот луч попал на сферу. Если луч не попадает на сферу, то метод возвращает $-1{,}0$.

```
type  Tsphere = class( Tfigure )
protected
radius : double;
procedure set_normal( point: Tvector ); override;
public
function  intersection( ray_origin    : Tvector;
                        ray_direction : Tvector ): double;
constructor  birth( center0: Tvector; color0: Tcvet;
                    n_spec0, v_spec0, radius0: double );
end;
```

В модуле `traceUnit` содержится процедура инициализации сцены init_scene (используется левая координатная система — см. рис. 4, картинная плоскость фиксирована и совпадает с плоскостью XY), задающая положение камеры, источника света, количество и параметры сфер и т. д., и *трассировщик* — рекурсивная функция trace, которая собственно и выполняет всю основную работу по трассировке лучей. Головной модуль `mainUnit` содержит процедуру отрисовки изображения, средства для поддержки управления программой и для сохранения трассированного изображения в файл. Использованный нами объектно-ориентированный метод построения программы даёт возможность легко и относительно просто дополнить сцену новыми объектами — любыми трёхмерными телами. Трассировщик.

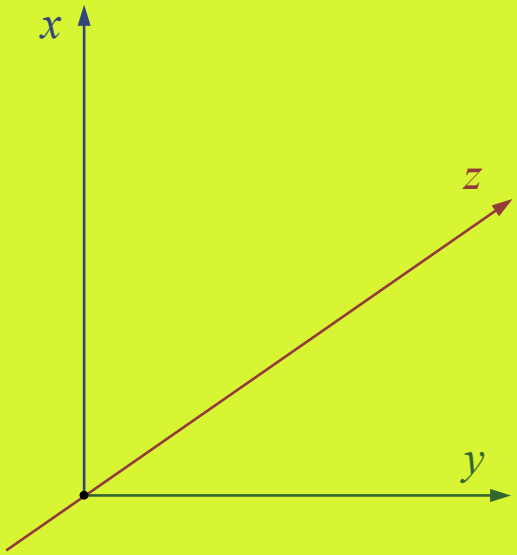


Рис. 4. Левая коорд. система

Прекрасным примером реализации идей рейтрессинга является программа POV-Ray (Persistence of Vision Raytracer). Эта программа свободно распространяется, причём с исходными кодами (на языке C++), снабжена отличной справочной документацией. На сайте разработчиков [7] можно также найти множество материалов по рейтрессингу.

Список литературы

1. В. И. Зенкин. Практический курс математического и компьютерного моделирования. Учеб. пособие. Калининград: изд. РГУ им. Канта, 2006.
2. В. В. Фаронов. Delphi 3. Учебный курс. М.: Нолидж, 1998.
3. Д. Тейлор, Дж. Мишель, Дж. Пенман, Т. Гоггин, Дж. Шемитц. Delphi 3: библиотека программиста. Спб, 1996.
4. А. А. Самарский. Введение в теорию разностных схем. М, 1974.
5. А. А. Самарский. Теория разностных схем. М, 1977.
6. Н. Н. Калиткин. Численные методы. М, 1978.
7. http://www.povray.org/