

# КОМПЬЮТЕРНАЯ ГРАФИКА

## ВВЕДЕНИЕ 3D ГРАФИКУ

© В. И. Зенкин, 2013

**Трёхмерная графика** (3D Graphics, 3 dimensions graphics) — раздел компьютерной графики, совокупность программных и аппаратных методов и инструментов предназначенных для изображения объёмных объектов на плоскости<sup>1</sup>.

Основные этапы получения трёхмерного изображения:

- **3D Моделирование** — построение математической модели *сцены* (3D scene), включающей:
  - трехмерные объекты, реальные или воображаемые, — геометрия;
  - виртуальные камеры (camera, viewpoint, eye);
  - освещение (lighting), источники света;
  - материалы (material) — информация о визуальных свойствах модели, таких как: цвет и характер поверхностей объектов (текстуры), отражение/преломление и т. д.;
  - специальные эффекты — имитация атмосферных явлений (туман, облака), пламя, дым, взрывы и пр.
- **Визуализация** (rendering, рендеринг) — построение изображения двумерной проекции сцены на *картинную плоскость* (view plane, image plane) в соответствии с математической моделью сцены.

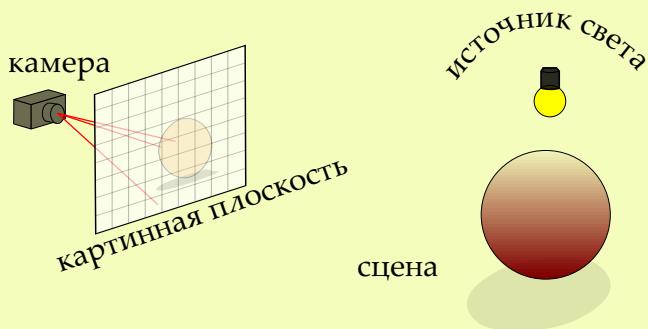
---

<sup>1</sup> Так как конечный результат — двумерное изображение, то термин «трехмерная графика» не совсем точен.



назад

закрыть



Для обработки модели в графической системе используют не весь объем информации об объекте, а только ту часть данных, которая определяет его геометрию: форму, размер и пространственное размещение. В соответствии с различными способами описания трехмерных объектов существуют разные способы их моделирования, которые условно можно разбить на две группы:

**Поверхностное (Boundary representation, B-rep)** — геометрические объекты сцены описываются при помощи задания набора его граничных поверхностей. Как правило, поверхностное представление используется в моделях, где нет необходимости обрабатывать каким-либо образом внутренность объектов.

**Объемное представление (Volume representation)** хранит информацию о любых, не обязательно видимых, частях объекта.

Поверхностное представление трехмерных объектов может задаваться:

- Неявной функцией (implicit function). Поверхность задается уравнением  $F(x, y, z) = 0$ , решения которого образуют поверхность объекта,



либо некоторое ее приближение. Например, для тора

$$\left( R - \sqrt{x^2 + y^2} \right)^2 + z^2 - r^2 = 0.$$

- Параметрически — координаты точек поверхности задаются как некоторые функции от двух параметров. Например, для тора

$$\begin{cases} x(\theta, \varphi) &= (R + r \cos \phi) \cos \theta, \\ y(\theta, \varphi) &= (R + r \cos \phi) \sin \theta, \\ z(\theta, \varphi) &= r \sin \theta, \end{cases} \quad 0 \leq \theta, \varphi \leq 2\pi.$$

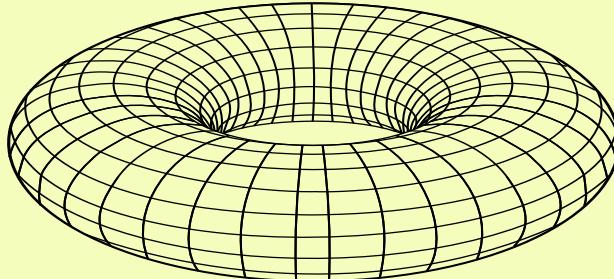


назад

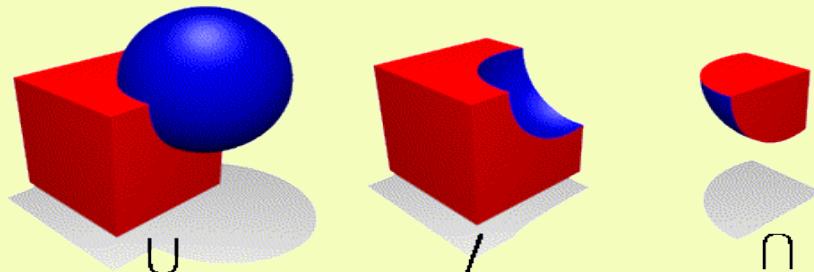
закрыть

или сплайнами — Безье, В-сплайнами, NURBS и др.

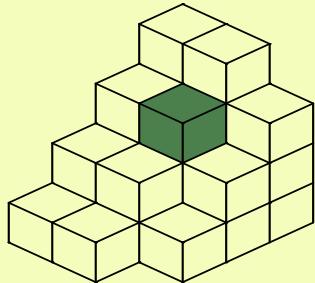
- Полигональными поверхностями. Поверхность (surface) в пространстве представляется в виде набора многоугольников (face).



- Методами конструктивной геометрии тел (Constructive Solid Geometry, CSG). Поверхность задается как результат применения последовательности различных множественных операций (объединение, пересечение, разности и т. д.) к геометрическим примитивам — параллелепипедам, шарам, конусам, пирамидам и т. д.



При объемном представлении трехмерного объекта требуется хранить информацию обо всех **вокселях** (voxel, от «volume element») геометрической



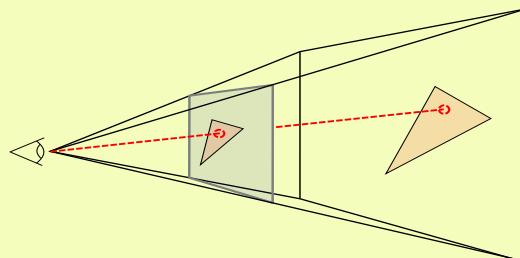
фигуры, как лежащих на поверхности объекта, так и внутренних. Каждый вокセル может иметь некоторое числовое значение, являющееся атрибутом соответствующей точки в пространстве. Серьезный недостаток этого метода: большой расход памяти. Например, для кубика с ребром 1024, при выделении одного байта на запись цвета вокселя, потребуется  $1024^3 \cdot 1$  байт = 1 Гб памяти.

Сократить объем данных можно, если учесть, что в 3D растре часто имеются области вокселов, имеющих одинаковые атрибуты, значения которых представляются в виде восьмеричного или двоичного дерева.

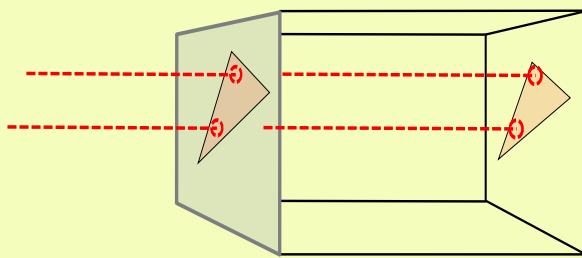
*Разреженное воксельное октодерево* (SVO – Sparse Voxel Octree). Корень дерева, является кубом, содержащим весь объект целиком. Каждый узел или имеет 8 потомков или ни одного. Эти 8 потомков формируют 8 регулярных подразбиения родительского узла. В результате всех подразбиений получается регулярная трёхмерная сетка вокселей, не все из которых содержат части объекта, поэтому и в созданном дереве такие воксели будут отсутствовать, то есть дерево будет разреженным. По некоторым оценкам SVO может стать одной из ключевых технологий хранения и представления данных в 3D графике реального времени в будущем.

**Камеры** это невизуализируемые объекты, которые можно разместить где угодно в составе трехмерной сцены. Они действуют подобно реальным фото- или видеокамерам, обеспечивая регистрацию изображений сцены с выбранной точки в соответствии с выбранным *типов проекции* 3D объектов на картинную плоскость.

Чаще всего применяются: а) перспективная (perspective) и б) ортографическая (orthographic) проекции.



а)



б)

Каждая камера задается своими:

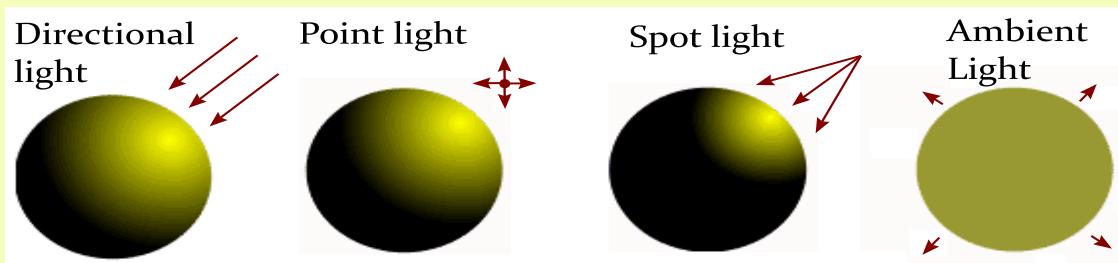
- Положением (location);
- Направлением (direction), которое может задаваться точкой, в которую направлена камера (target);
- Типом проекции (types of projection), в зависимости от которого для камеры могут определяться дополнительные параметры.

Правильное настроенное освещение является одним из наиболее существенных факторов обеспечения реализма сцены. Оно создает контрастность объектов, делает использованные материалы более яркими и выразительными и позволяет настраивать тени объектов.

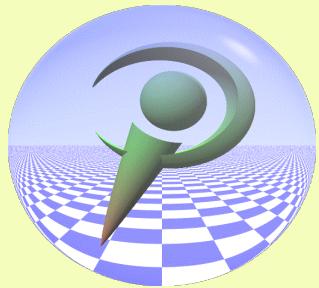
Источники света, в соответствии с назначением, различаются по типам, наиболее употребительными из которых являются:

- Directional light — источник параллельных лучей.
- Omni light (Point light), всенаправленный — лучи распространяются равномерно во всех направлениях.
- Spot light — прожектор, источник конически расходящихся лучей.
- Ambient Light, рассеянный свет — общий фоновый, не исходящий от какого-либо конкретного источника света.

В разных программах трехмерной графики и визуализации существует множество других типов источников света, отличающиеся по своему функциональному предназначению.



Далее описание трехмерных сцен будем приводить на языке описания сцен SDL (Scene Description Language), применяющимся в программе POV-Ray.



POV-Ray (Persistence of Vision Raytracer) — программа, генерирующая изображения методом *трассировки лучей* (ray tracing), бесплатное свободное<sup>2</sup> программное обеспечение с доступным исходным кодом [5]. Трассировка лучей — это метод, позволяющий создавать фотorealистические изображения любых трехмерных сцен. Фотorealизм достигается за

счет адекватных математических моделей оптических свойств света и его взаимодействия с объектами. Основные возможности POV-Ray:

1. Язык высокого уровня SDL для описания 3D-сцен.
2. Поддержка большого количества примитивов для составления других тел методами конструктивной геометрии CSG.
3. Возможность добавления атмосферных эффектов, инструменты отражения, преломления, каустики и многое другое.
4. Наличие полного руководства пользователя. Поддержка программы третьими сторонами. В Интернете можно найти большое количество инструментов, текстур, моделей, сцен и руководств по POV-Ray.
5. Изображения до 48 бит на пикセル. Фотorealистичный рендеринг.

---

<sup>2</sup>Лицензия AGPLv3: создание, использование, воспроизведение, распространение, модификация.



◀◀

▶▶

◀

▶

назад

закрыть



AUTOBAHN - JVP 2005 - MegaPOV 1.1 - 2246"

◀◀

▶▶

◀

▶

назад

закрыть



Copyright 2000 Gilles Tran

Gilles Tran © 2002 www.oyonale.com



назад

закрыть



Copyright © 2005 Rene Bui



назад

закрыть



Copyright 1999 Johannes Ewers



назад

закрыть



◀◀

▶▶

◀

▶

назад

закрыть

POV-Ray не имеет собственных встроенных средств интерактивного 3D моделирования: есть множество программ, экспортирующих в его формат (в том числе популярная бесплатная программа Blender). Сцены для трассировки описываются в текстовых файлах на C-подобном языке описания сцен SDL. Язык содержит идентификаторы, зарезервированные слова, вещественные числа, строки, спецсимволы и комментарии (однострочные // и многострочные /\* \*/). SDL-файлы являются входными данными для трассировщика, а результат его работы — растровое изображение, которое можно сохранить в различных разрешениях и форматах.

Сцены POV-Ray состоят из объектов вида:

тип\_объекта { параметры }

Параметры бывают обязательные и дополнительные. Обязательные необходимо указывать при создании объекта, сразу после открывающей фигурной скобки. Обязательные параметры могут быть числами или векторами, имеют фиксированный порядок и разделяются запятыми. Дополнительные параметры требуют указания имени параметра, за которым следует его значение (число, вектор или объект), запятыми они не разделяются.

Имена переменных, объектов и т. д. объявляются так

#declare идентификатор = выражение;

Арифметические и логические операторы SDL аналогичны тем, что используются в языке C.



назад

закрыть

Векторы используются для обозначения координат и цветов; компоненты векторов заключаются в ломаные скобки и разделяются запятыми.

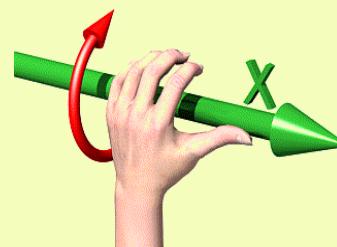
`<0.2, 0.4, 0.9, 0.1>`

`x`

// сокращение `<1, 0, 0>`

В POV-Ray существует два основных способа моделирования: средствами конструктивной геометрии с помощью объединения примитивов (сфер, цилиндров, кубов и т. д.) в более сложные объекты, и путем создания

собственных функций, описывающих сложную поверхность. Полигональные объекты также можно использовать в сценах, но они служат только для импорта моделей из сторонних приложений. POV-Ray по умолчанию использует левую прямоугольную декартову систему координат XYZ, где оси X и Y параллельны экрану дисплея, а ось Z направлена внутрь.



Включение (`include`) внешних файлов, которые могут содержать предопределенные цвета, материалы, текстуры, элементы сцен и т. д.:

```
#include "colors.inc"      // предопределенные цвета
#include "stones.inc"     // предопределенные элементы
#include "textures.inc"    // предопределенные текстуры
```



назад

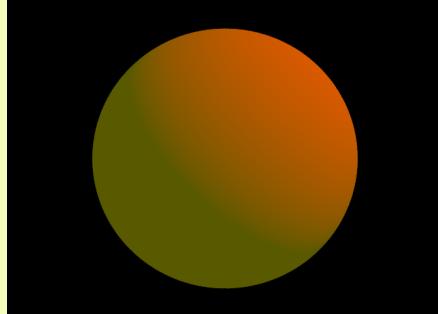
закрыть

```

#include "colors.inc"           // предопределенные цвета
camera {                      // камера:
    location < 0, 0, -4 >     // положение камеры
    look_at < 0, 0, 0 >        // направление камеры
}
light_source {                 // источник света:
    < 4, 4, -3 >            // положение
    color Magenta             // цвет
}
sphere {                       // объект сфера
    < 0, 0, 0 >, 1.5         // положение и радиус
    pigment { color Yellow }   // цвет
}

```

Для объекта Сфера ( *sphere* ) указаны два обязательных параметра: положение (вектор) и радиус (число) и объект «пигмент» ( *pigment* ), вложенный в сферу. Пигменты задают цвет поверхности объекта-родителя, который не обязательно является «отдельным» цветом, это может быть узор — *pattern* , список цветов — *color List* , карта цветов — *color map* и др. Пигмент определяет цвет поверхности материала без учета освещения модели, образующихся при этом бликов и отражений. Поскольку тип проекции для камеры явно не указан, используется тип по умолчанию — перспективный.



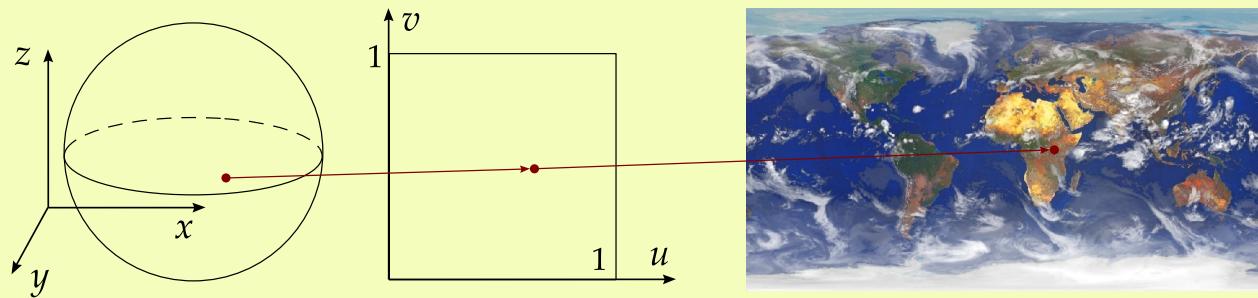

назад

закрыть

**Текстура (texture)** — изображение, накладываемое на поверхность 3D объекта, для придания ему цвета, окраски и/или иллюзии рельефа. Применение текстур позволяет, в частности, воспроизвести мелкие детали поверхности объекта, создание которых другими методами было бы чрезмерно ресурсозатратным.

Чаще всего используются *растровые* и *вычисляемые* текстуры. Растровой текстурой называется текстура, сформированная из растровых изображений, вычисляемая текстура определяется с помощью какого-либо математического алгоритма.

Процесс наложения текстур (*texture mapping*) заключается в построении *uv-преобразования* (*uv тар*), отображающего координаты на поверхности трёхмерного объекта ( $x, y, z$ ) в точки единичного квадрата, координаты которого по традиции обозначают  $(u, v)$ ,  $0 \leq u, v \leq 1$ . Затем координатам  $(u, v)$  ставится в соответствие значения цвета пикселов текстуры — *текселов* (*TEXture ELelement, texel*).



```
#include "colors.inc"
camera {
    location < 1, 1, -4 >           // камера
    look_at   < 0, 0, 0 >            // положение камеры
}
light_source {
    < 4, 4, -3 >                 // светильник
    color Yellow                   // положение источника света
}
sphere {
    < 0, 0, 0 >, 1.5             // объект сфера
    pigment {                      // центр сферы и радиус
        image_map { jpeg "Earth.jpg" map_type 1 }
    }
    rotate < 0, -60, 0 >          // текстура
}                                     // поворот сферы вокруг оси Y
```

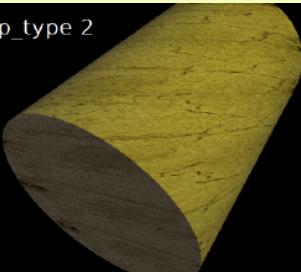
Отображение  $(x, y, z) \rightarrow (u, v)$  задается в зависимости от типа 3D поверхности, наиболее часто используют планарную (плоскую), сферическую

и цилиндрическую проекции. В POV-Ray для этого предусмотрен параметр `map_type`, значения которого — 0 (планарная, по умолчанию), 1 (сферическая), 2 (цилиндрическая), 5 (проекция на тор). Значение параметра `jpeg` задает путь к растровому файлу текстуры — изображение на предыдущем слайде.

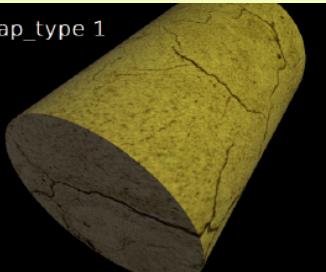




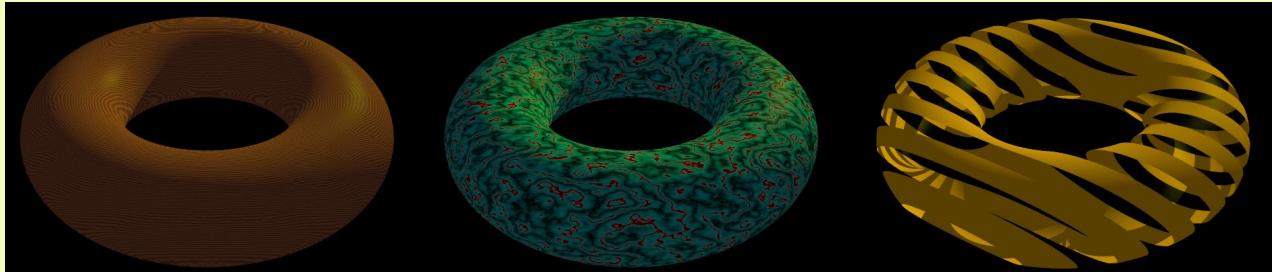
map\_type 2



map\_type 1



```
#include "colors.inc"
camera {
    location < 0, 1, -7.5 >           // камера:
    look_at   < 0, 0, 0 >             // положение камеры
}
light_source {
    < 4, 4, -5 >                   // осветитель:
    color Yellow                     // направление камеры
}
cylinder {
    < -1, -1, -2 >,               // цилиндр:
    < 2, 2, 2 >,                 // центр основания цилиндра
    2.0                            // центр другого основания
    pigment {                      // радиус оснований цилиндра
        //image_map { jpeg "23_texture.jpg" map_type 1 }
        image_map { jpeg "23_texture.jpg" map_type 2 }
    }
}
```

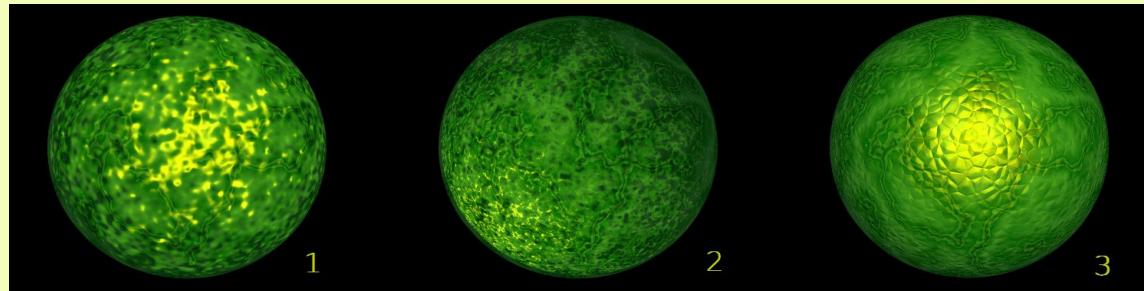


```
#include "textures.inc"
camera {
    location < 0, -7.5, -16.5 >           // камера:
    look_at   < 0, 0, 0 >                     // положение камеры
    angle     40                                // направление камеры
                                                // угол
}
light_source {
    < 14, 20, -1 >                         // осветитель:
    color < 0, 1, 1 >                      // положение источника света
                                                // цвет источника света
}
torus {
    4, 1.5                                     // тор
    rotate -60*x                               // большой и малый радиусы
    pigment { //DMFWood3                       // поворот относительно Ох
        //Blood_Marble                         // текстуры:
        Peel                                    // определены в "textures.inc"
        scale <1.7, 0.9, 1>                  // peel – шелуха
                                                // масштабирование
    }
}
```

Одним из методов процедурного текстурирования является *рельефное текстурирование* (*relief mapping*). В частности, процедура *bump<sup>3</sup> mapping* заключается в случайном отклонении нормалей каждого пикселя от «истинной» нормали, рассчитанной для соответствующих точек поверхности с целью имитации её неровностей. К примеру, поверхности нефритовой сферы придаются искусственные неровности разных видов.

```
sphere {  
    < 0, 0, 0 >, 1.5  
    pigment { Jade }  
    //normal { bumps 1 scale 0.06 }  
    //normal { wrinkles 2 scale 0.06 }  
    normal { crackle 1 scale 0.09 }  
    finish { specular 1 }  
}
```

// объект сфера  
// центр сферы и радиус  
// нефрит, в "textures.inc"  
// 1. bump  
// 2. wrinkle – морщина  
// 3. crackle – трещина



<sup>3</sup>Вимп — выпуклость, шишка.

Для большего реализма объектов трехмерной сцены и всей сцены в целом необходимо учитывать параметры, относящиеся к взаимодействию поверхностей с падающим на них светом, как-то:

- Рассеянное освещение, учитывающее не только свет, исходящий от источников, но и отраженный от объектов, освещенных прямым светом.
- Степень прозрачности объекта.
- Диффузное отражение — свет, который поверхность объекта рассеивает, окрашивая его в цвет своей поверхности, и отражает назад. Яркость диффузного отражения зависит от угла между нормалью поверхности и направлением падения лучей света.
- Блики, которые образуются на очень гладких поверхностях, когда лучи света сразу отражают назад, прямо в камеру. Так как рассеивания света при этом почти не происходит, то на поверхности образуется яркое пятно, соответствующее цвету источника света.
- Зеркальное отражение и преломление.

В POV-Ray рассеянное освещение (radiosity) включается в общих настройках сцены:

```
global_settings {  
    radiosity { count 500 minimum_reuse 0.018 brightness 0.8 }  
}
```



назад

закрыть

Подробное описание методов radiosity и значения параметров см. в документации к POV-Ray. В связи с большим количеством дополнительных расчётов, использование механизма рассеянного освещения в сложных сценах может привести к существенному замедлению рендера.

Прозрачность объекта можно задать при помощи модификатора filter, значения которого могут меняться от 0 (полностью непрозрачный объект) до 1 (на 100% прозрачный). Остальные перечисленные параметры определяются в разделе finish. Например

```
finish {
    diffuse    0.8      // 80% света, падающего на
                        // поверхность, рассеивается;
    ambient    0.05     // яркость неосвещенных мест сцены
                        // увеличена на 5%;
    phong      0.13     // в блик уходит 13% энергии света;
    phong_size 160      // размер блика, чем более гладкая поверхность,
                        // тем больше должна быть эта величина;
    reflection 0.2      // коэффициент отражения поверхности;
    refraction 1         // учитывать преломление лучей;
    ior        1.0333    // коэффициент преломления
                        // (ior -- Index Of Refraction);
}
```

Рассмотрим простую сцену: стеклянный шар на фоне текстурированной плоскости.



назад

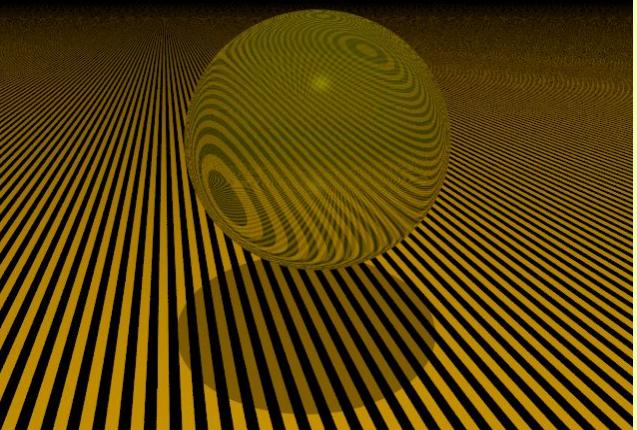
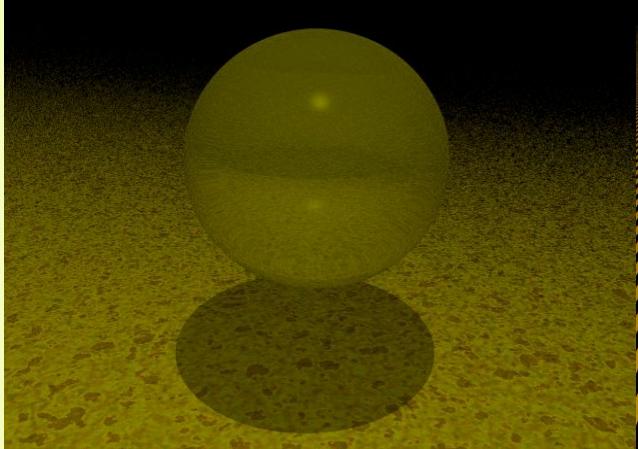
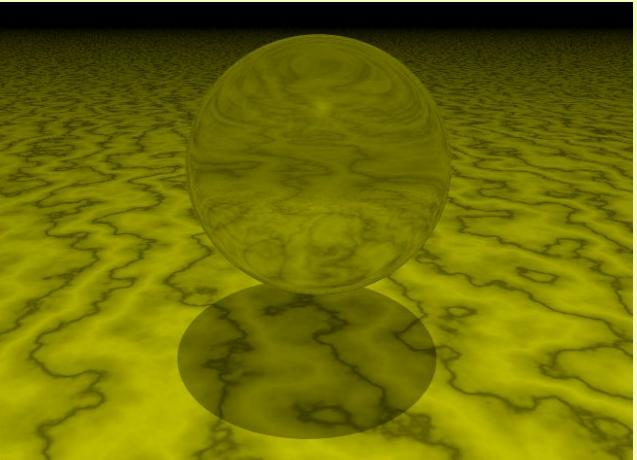
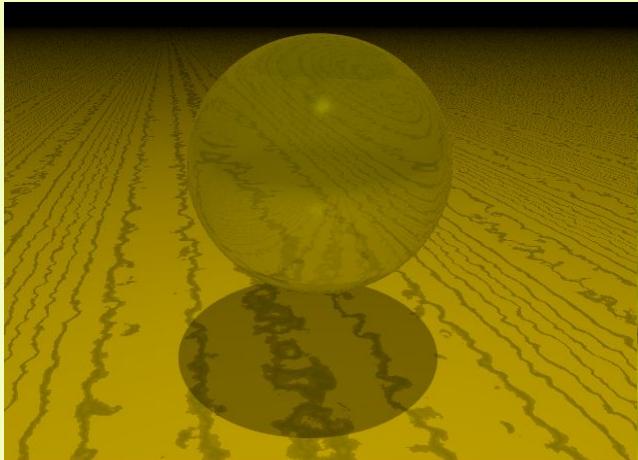
закрыть

```
#include "colors.inc"
#include "textures.inc"
global_settings { radiosity { count 1000 minimum_reuse 0.018 brightness 1 } }
camera {
    location < -2, 2, -8.0 >           // камера:
    look_at   < 0, -1, 0 >             // положение камеры
    angle     58                      // направление камеры
    angle     58                      // угол обзора камеры
}
light_source {
    < -1, 10, -6 > color Yellow      // осветитель:
    < -1, 10, -6 > color Yellow      // положение и цвет источника
}
plane{
    < 0, 1, 0 >, -5.0                // объект плоскость:
    //texture{ Tan_Wood scale 0.6 }    // нормаль и расстояние от O
    //texture{ White_Marble scale 2.6 } // текстуры
    //texture{ Rusty_Iron scale 1.3}    // определены
    texture{ Peel scale 0.3}          // в "textures.inc"
}
sphere {
    < 0, 0, 0 >, 1.9                // объект сфера:
    pigment { color White*0.7        // центр сферы и радиус
              filter 0.9             // цвет сферы
              filter 0.9             // сфера прозрачна на 90%
}
    finish { diffuse    0.8 ambient   0.05 phong      0.13
              phong_size 160 reflection 0.2 refraction 1 ior 1.033
}
}
```



назад

закрыть



◀◀

▶▶

◀

▶

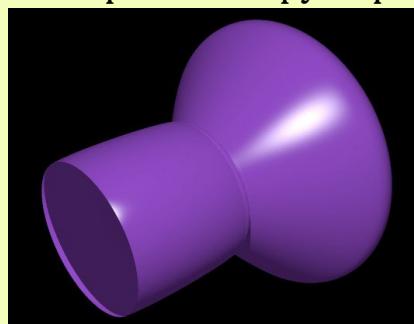
назад

закрыть

Наряду со «стандартными» примитивами, типа сферы, плоскости, параллелепипеда, конуса и т. д. в POV-Ray предусмотрены другие методы определения объектов.

*Lathe* (*токарный станок*) — объект, получаемый при вращении двумерной кривой вокруг прямой. Эта кривая определена рядом точек, связанных линейным (*linear\_spline*), квадратным (*quadratic\_spline*), кубическим (*cubic\_spline*) или сплайном Бэзье (*bezier\_spline*). Кривая автоматически не замкнута, если нужно получить замкнутый контур, то надо задать последнюю точку, равной первой. Метод вращения сплайн-кривой для получения поверхности является стандартным для многих программ 3D моделирования.

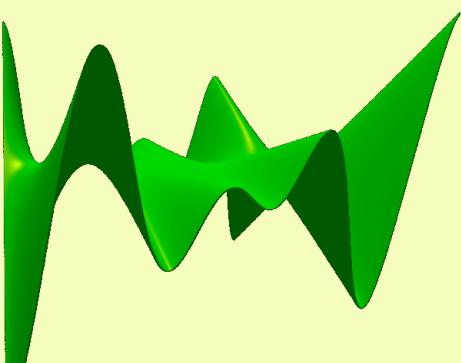
```
lathe {  
    cubic_spline  
    7,  
    < 2, 0 >, < 3, 0 >, < 3, 5 >, < 2, 5 >,  
    < 6, 9 >, < 5, 9 >, < 2, 0 >  
    texture {  
        pigment { color rgb < 0.4, 0.1, 0.8 > }  
        finish { phong 1 reflection 0.2 }  
    }  
}
```



## Параметрическое задание поверхности.

```
parametric {
    function{ u }                                // x(u,v)
    function{ u*v*cos(u*3*pi) }                  // y(u,v)
    function{ v }                                // z(u,v)
    < -5, -2 >, < 5, 2 >                      // границы u, v
    max_gradient 9
    accuracy      0.00035
    precompute    18 x,y,z
    texture {
        pigment{ color Green}
        finish { phong 0.5 }
    }
}
```

По очевидным причинам для этого способа рендер идет медленно.

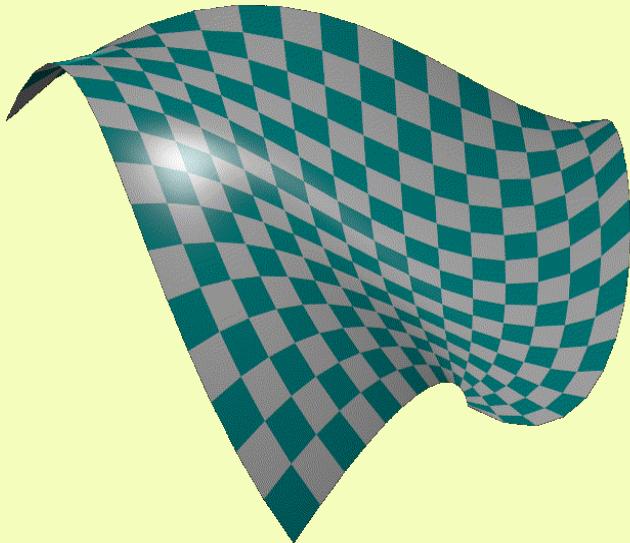


*Кусочное моделирование (patch<sup>4</sup> modeling).* Поверхность задается кусками, форма которых определяется многоугольниками или сплайнами по кон-

трольным (управляющим) точкам. POV-Ray не трассирует патчи напрямую, а использует аппроксимацию бикубическими поверхностями Безье. Сложные кусочные модели обычно создают в специальных программах, таких как JPatch (<http://www.jpatch.com/>), POVLAB (<http://pdelagrange.free.fr/povlab/>) и других; многие из этих программ могут

экспортировать свои объекты в формат POV-Ray. В примере ниже точки B11, B14, B41, B44 определяют углы патча, остальные точки — управляющие. Изменяя их положение можно произвольно менять форму поверхности. Для гладкого соединения нескольких патчей используются те же приемы, что применялись для сочленения 2D сплайнов Безье.

<sup>4</sup> Patch — заплатка, лоскут, обрывок.



```
#include "textures.inc"
camera { location <1.6,5,-6> look_at <1.5,0,1.5> angle 30 }
light_source { < 0, 10, -5 > rgb 1 }
#declare B11 = < -1, 0, 3 >; #declare B12 = < 1, 2, 3 >;
#declare B13 = < 2, 0, 3 >; #declare B14 = < 3, 0, 3 >; // строка 1
#declare B21 = < 0, 2, 2 >; #declare B22 = < 1, 0, 2 >;
#declare B23 = < 2, 0, -2 >; #declare B24 = < 3, 1, 2 >; // строка 2
#declare B31 = < 0, 0, 1 >; #declare B32 = < 1, 0, 2 >;
#declare B33 = < 2, 0, 1 >; #declare B34 = < 3, 0, -1 >; // строка 3
#declare B41 = < 1, -2, 1 >; #declare B42 = < 2, 0, 1 >;
#declare B43 = < 2, 0, 0 >; #declare B44 = < 2, 0, 0 >; // строка 4
bicubic_patch {
    type 1 flatness 0.001 u_steps 4 v_steps 4
    uv_vectors < 0, 0 > < 1, 0 > < 1, 1 > < 0, 1 >
    B11, B12, B13, B14
    B21, B22, B23, B24
    B31, B32, B33, B34
    B41, B42, B43, B44
    uv_mapping
    texture {
        pigment {
            checker color rgbf < 1, 1, 1, 0.5 >
            color rgbf < 0, 0.8, 0.8, 0.7 >
            scale 1/16
        }
        finish { phong 0.6 phong_size 20 }
    }
}
```



назад

закрыть

Чайник *Юта*<sup>5</sup> (*The Utah Teapot*) состоит из 32 кусков бикубической поверхности Безье, координаты опорных точек которых и являются исходным описанием модели чайника. Точки образуют массив из 306 элементов. Корпус чайника образован из 12 кусков, ручка — из четырёх, остальные части формируют носик и крышку чайника.

Определения для Безье-патчей модели чайника содержится в файле teapot.inc, входящего в поставку POV-Ray.

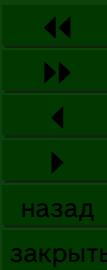


<sup>5</sup> Модель чайника Юта или чайника Ньюэлла была создана в 1975 году Мартином Ньюэллом (Martin Newell) в университете Юты для исследований в области компьютерной графики, для которых ему была нужна модель всем известного реального не слишком геометрически сложного объекта. С тех пор эта модель часто используется в 3D графике в качестве объекта для тестов: в Autodesk 3ds Max чайник Юта считается графическим примитивом, в библиотеках DirectX и OpenGL есть специальная функция Create Teapot и т. д. Сейчас в трехмерной графике сцены с чайником Ньюэлла являются чем-то вроде аналога «Hello, World» в программировании. Физический оригинал заварочного чайника Ньюэлла бережно хранится в музее компьютерной истории (Computer History Museum, <http://www.computerhistory.org/collections/catalog/X398.84> ).

*Полигональная сетка (polygon mesh)* — множество многоугольников, вершины, рёбра и грани, которых в совокупности определяют форму 3D объекта. Гранями обычно являются треугольники, четырехугольники или другие простые выпуклые полигоны.

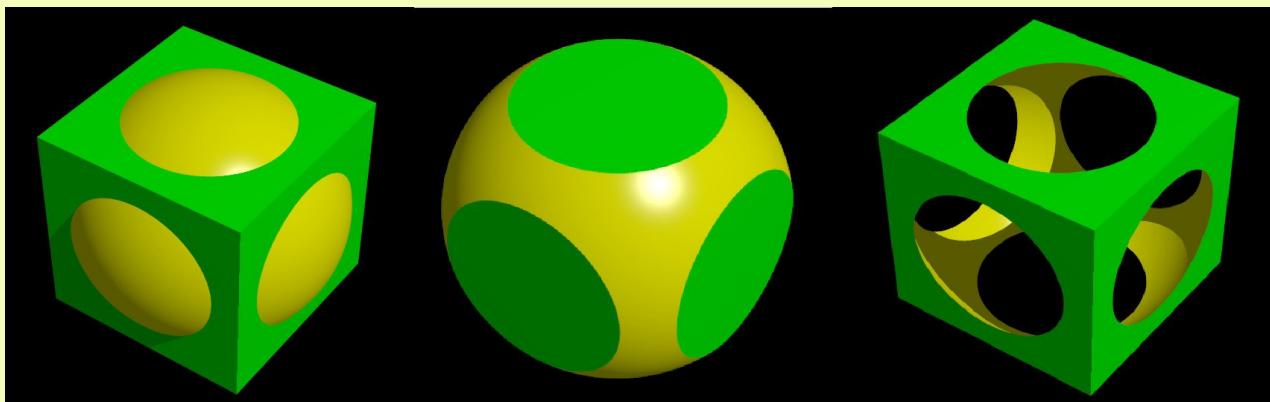
В POV-Ray объект mesh является множеством треугольников, которые задаются координатами своих вершин. Каждому из треугольников можно назначить отдельную текстуру:

Также имеется «усовершенствованный» объект mesh2, лучше соответствующий внутреннему представлению сеток в POV-Ray, и потому быстрее визуализирующийся рендером. Объекты типа mesh2 предназначены в основном для сторонних моделлеров, имеющих возможность экспорта в формат POV-Ray, а не для «ручного» конструирования.



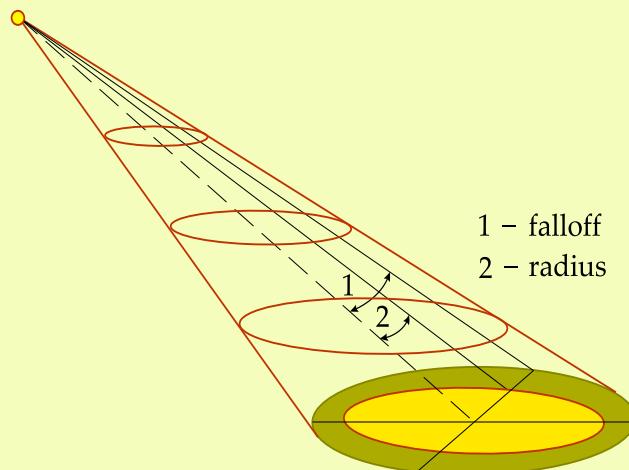
## Методы конструктивной геометрии (CSG, Constructive Solid Geometry).

```
#include "colors.inc"
camera { location < 5, 8, -7 > look_at < 0, 0, 0 > angle 25 }
light_source { < 14, 18, -3 > color White }
difference {
    box { < -1, -1, -1 >, < 1, 1, 1 >
        pigment {color Green}
        finish { phong 1}
    }
    sphere { <0.0, 0, 0> 1.24
        pigment {color Yellow}
        finish { phong 1}
    }
}
```



Источники света в реальности не являются объектами трехмерной сцены, но формально в POV-Ray они считаются таковыми. Трассировщик поддерживает различные типы источников света. На сцене могут быть несколько осветителей разного типа, которые можно группировать вместе и/или с другими объектами. Основные типы осветителей:

- Point Lights — точечный источник света, задан по умолчанию.
- Parallel Lights — имитация сильно удаленного осветителя.
- Spotlight — прожектор. Дополнительные параметры: углы falloff, radius,



а также tightness (плотность, непроницаемость), управляющие затуханием освещенности к границе конуса.

Угол falloff определяет общий световой конус, а radius — «активную» (hot-spot) его часть. Параметр tightness задает дополнительное смягчение границ светового конуса. Если интенсивность источника света равна  $I$ , то интенсивность освещения участка с углом  $\alpha$  по отношению к центральной линии светового конуса  $I \cos \alpha \cdot \text{tightness}$ .

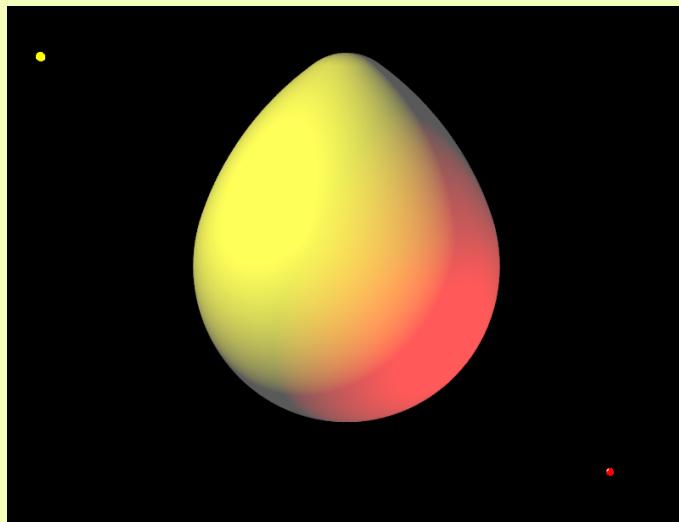
```
#include "colors.inc"
camera { location < -1, 3, -10 > look_at < 0, 0, 0 > angle 40 }
#declare light_coord_1 = < -2.5, 2.5, -3 >; // коорд. осветителя #1
#declare light_coord_2 = < 2.0, -1.0, -3 >; // коорд. осветителя #2
light_source { // источник света #1
    light_coord_1 color Yellow*1.8
    spotlight
    radius      100
    falloff     130
    tightness   1.4
}
sphere      { // прозрачная сфера -- имитация источника света #1
    light_coord_1, 0.03
    pigment { color Yellow filter 0.999 }
    finish   { reflection 10.05 }
}
light_source { // источник света #2
    light_coord_2 color Red*2
    spotlight
    radius      130
    falloff     150
    tightness   0.6
}
```



назад

закрыть

```
sphere    { // прозрачная сфера -- имитация источника света #2
    light_coord_2, 0.03
    pigment { color Red filter 0.999 }
    finish   { reflection 10.05 }
}
ovus     { 1.70, 0.6
    texture{
        pigment{ color White }
        finish { phong 0.05 }
    }
}
```

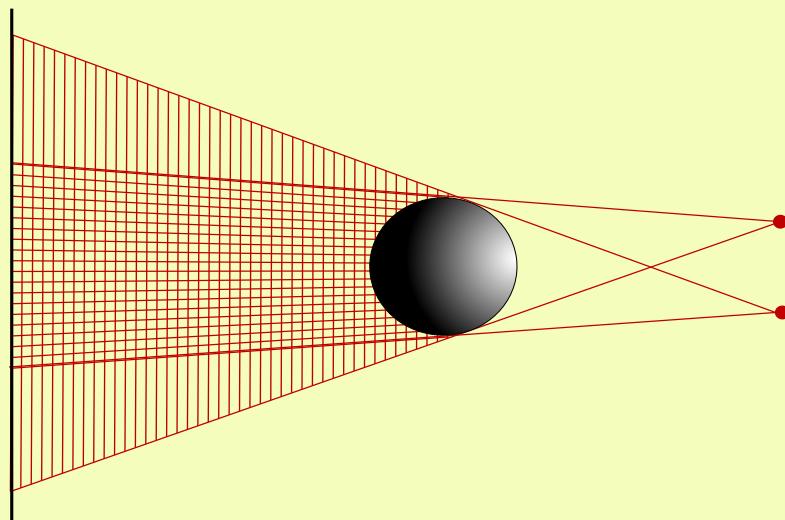


## Другие типы источников света:

- **Цилиндрический.** Параметры те, что и для spotlight. В примере выше замените ключевое слово spotlight на cylinder.
- **Фоновое освещение (ambient)** задается в глобальных настройках сцены, например

```
global_settings { ambient_light rgb <1, 0, 0> }
```

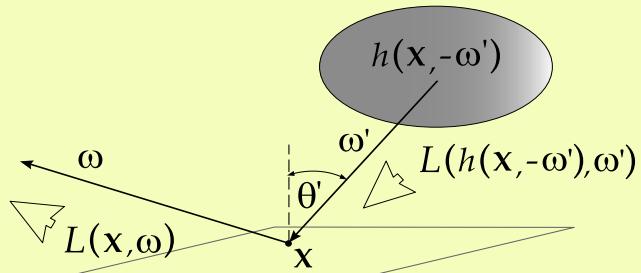
- **Area Light Source** — несколько осветителей, сгруппированных в некоторой области. Такое освещение позволяет получить более реалистичные мягкие переходы теней — см. документацию к POV-Ray.



**Визуализация** (rendering, рендеринг) трехмерной сцены в настоящее время осуществляется с помощью различных математических алгоритмов и компьютерных технологий.

Математически реалистичные алгоритмы визуализации в той или иной степени основаны на решении *уравнения рендеринга*, которое выражает закон сохранения энергии.

Для излучательности<sup>6</sup>  $L(x, \omega)$  в точке поверхности  $x$  по направлению  $\omega$ , уравнение рендеринга имеет вид



$$L(x, \omega) = L_e(x, \omega) + \int_{\Omega} L(h(x, -\omega'), \omega') f_r(\omega', x, \omega) \cos \theta' d\omega', \quad (1)$$

где  $L_e$  — излучательность точки поверхности от прямого освещения, а интеграл по полусфере входящих направлений света  $\Omega$  описывает взаимодействие света, пришедшего от других освещенных поверхностей, с учетом функции видимости точек сцены  $h$ , функция двунаправленного отражения (преломления)  $f_r$  задает оптические свойства поверхности.

Линейное интегральное уравнение рендеринга определяет количество светового излучения в определённом направлении как сумму собственного и отражённого излучения [9].

<sup>6</sup>Излучательность — отношение потока излучения, исходящего от малого элемента излучающей поверхности к площади этого элемента.



назад

закрыть

Программное и аппаратное обеспечение может использовать различные комбинации алгоритмов для рендеринга. Можно выделить следующие методы визуализации

- *Алгоритмы построчной развертки (scanline rendering).* Через каждую точку сцены проводится воображаемая сканирующая линия и вычисляются, какие полигоны лежат на пути этой линии, определяя необходимый цвет пикселя на картинной плоскости, в зависимости от того, какие текстуры и цвета были назначены полигонам, встретившимся на пути скан-линии. Основной проблемой этого метода является отделение видимых полигонов от невидимых. Чтобы решить эту проблему используется т. н. *Z-буфер*. Рендер определяет все полигоны, лежащие на пути скан-линии и назначает каждому полигону Z-значение в зависимости от его удаленности от картинной плоскости. При визуализации обсчитываются только полигоны с наименьшим Z-значением, остальные отбрасываются. Рендереры, основанные на алгоритме scanline, созданы в первую очередь для скорости, а не для качества. Они часто применяются в играх, различных симуляторах реального времени или в пакетах 3D моделирования.
- *Рейкастинг (ray casting, метод «бросания лучей»)* — лучи испускаются из камеры, один луч на каждый пиксель, и находится ближайший к камере объект, который блокирует путь распространения этого луча. Рендеринг сцены строится на основе расчетов точек пересечения



назад

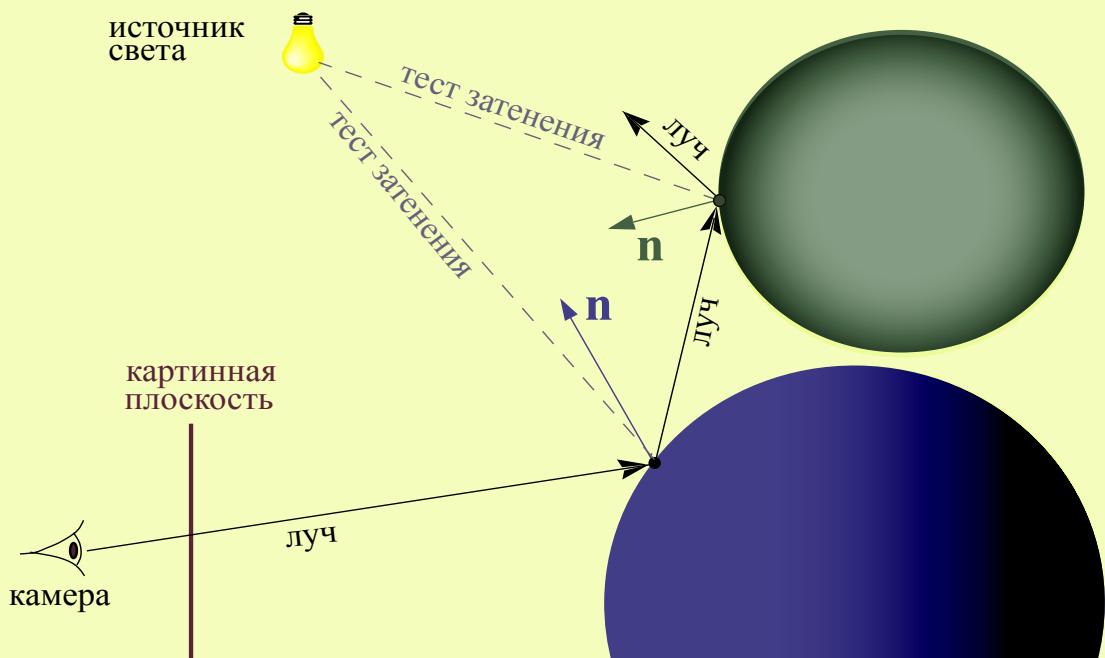
закрыть

лучей с визуализируемой поверхностью. На основе алгоритма рейкастинга визуализация сцены осуществляется очень быстро, что позволяет применять её в реальном времени. Компания Id Software применяла его своих игр ещё в начале 90-ых гг XX в. В частности, рейкастинг использовался в игре *Wolfenstein 3D* (1992 г, операционная система DOS, разрешение  $320 \times 200$ , 256 цветов, спрайтовая графика).



- Трассировка лучей (*ray tracing*) — так же, как в рейкастинге, из камеры на объекты сцены направляются лучи, с помощью которых определяется цвет пикселя на картинной плоскости, но при этом луч не прекращает своё распространение, а разделяется на три луча-компоненты, каждый из которых вносит свой вклад в цвет пикселя: отражённый, теневой и преломлённый. Путь каждого из этих лучей

также отслеживается до следующего пересечения с поверхностью. Количество таких пересечений задает глубину трассировки, что определяет качество и фотorealистичность изображения. Метод трассировки лучей позволяет получить вполне фотorealистичные изображения, однако из-за большой ресурсоёмкости процесс визуализации занимает значительное время. Именно этот метод применяется в POV-Ray.



## Список литературы

1. Марк Джамбруно. Трехмерная (3D) графика и анимация. 2-е изд. М.: Вильямс, 2002.
2. Роджерс Д., Адамс Дж. Математические основы машинной графики. М.: Мир, 2001.
3. Виктор Порев. Компьютерная графика. СПб.: БХВ-Петербург, 2004.
4. В. П. Иванов, А. С. Батраков. Трехмерная компьютерная графика. М.: Радио и связь, 1995.
5. Сайт программы POV-Ray ([www.povray.org](http://www.povray.org)).
6. <http://www.render.ru/>
7. <http://3d-uroki.org/>
8. <http://www.gamedev.net/>
9. Kajiya, James T. (1986), The rendering equation, Siggraph 1986: 143, doi:10.1145/15922.15902, ISBN 0-89791-196-2

&lt;&lt;

&gt;&gt;

◀

▶

назад

закрыть