# Язык представления трехмерных объектов VRML

Язык VRML (Virtual Realty Modelling Languagy) предназначен для описания трехмерных изображений и оперирует объектами, описывающими геометрические фигуры и их расположение в пространстве. Язык был создан в 1994 году консорциумом во главе с Silicon Graphics для применения в сетях INTERNET.

Действующим стандартом является VRML 97, известный как ISO/IEC 1477.

Версий языка на данный момент три.

- Первая попытка сделать сеть трехмерной родила версию 1.0.
- Версия 2.0 появилась позже и имела намного большие возможности. Именно эту версию языка разработчики отправили на стандартизацию в ISO/IEC.
- После этого была разработана новая спецификация языка VRML 97. Она мало отличается от 2.0 и содержит лишь небольшие изменения и поправки.

Vrml-файл представляет собой обычный текстовый файл в формате ASCII с расширением .wrl, интерпретируемый браузером.

VRML дает разработчику возможность не только проектировать статические и динамические 3D модели, он также позволяет включать и обрабатывать в этих моделях гиперссылки на звуковые, видео, html-файлы и другие VRML объекты.

Поскольку большинство браузеров не имеет встроенных средств поддержки VRML, для просмотра vrml-документов необходимо подключить вспомогательную программу – vrml - браузер, например.

Как и в случае с HTML, один и тот же vrml-документ может выглядеть по-разному в разных vrml - браузерах.

Основные VRML браузеры:

- Cosmo Player (Cosmo Software)
- Cortona VRML client (ParallelGraphics)
- WorldView (Intervista)

Наиболее известны Cosmo Player и WorldView. Последние версии Cosmo Player и WorldView-браузеров появились на свет в 1997-98 годах. Они более не развиваются и не поддерживаются.

В лекции используется кроссплатформенный (MS Windows, Linux и Mac OS X) VRML/X3D - браузера view3dscene v.3.15.0. (30.12.14)

http://vrml.org.ru/htm/news/index.html?/htm/news/news.html

Существует немало VRML-редакторов, делающих удобней и быстрее процесс создания vrml-документов, однако несложные модели, рассматриваемые в данной статье, можно создать при помощи самого простого текстового редактора.

# История VRML

В январе 1994 года, Mark Pesce и Tony Parisi придумывают концепцию трехмерного HTML, своеобразного языка описания трехмерных сцен с поддержкой гиперссылок, и создают пакет программ и экспериментальный формат, названные ими Labyrinth - первый прообраз VRML.

Весной 1994 года на первой ежегодной конференции по World Wide Web в Женеве участники конференции согласились иметь общий язык для определения 3-мерного описания и гиперсвязей для всемирной паутины — аналога HTML для виртуальной действительности. Так впервые возник термин Virtual Reality Markup Language (VRML), где слово «Markup» вскоре было заменено на «Modelling» чтобы отразить графическую природу VRML (в литературе встречаются и другие варианты расшифровки, в частности Virtual Reality Meta Language, однако сочетание Virtual Reality Modelling Language является не только наиболее устоявшимся.

Были также сформулированы основные требования, которым он должен был удовлетворять: независимость от компьютерной платформы, расширяемость и возможность работы по низкоскоростным каналам связи.

Вскоре Silicon Graphics публично заявила об открытости технологии и анонсировала разработку средств просмотра.

Выразительные возможности языка VRML 1.0, были крайне бедны. С его помощью можно было описывать только полигональные статические сцены с гиперссылками.

В 1996 г. международным vrml-сообществом был объявлен конкурс на лучший вариант спецификации следующей версии языка, получившей название VRML 2.0 и утвержденной в августе 1996 г.

Самым главным отличием, помимо изменения структуры дерева трехмерной сцены и добавления большого количества новых узлов, стало введение в язык средств динамического изменения сцены и интерактивного взаимодействия с пользователем. Самым главным новшеством была возможность моделировать не только внешний вид объектов, но и программировать сколь угодно сложное их поведение. Появились возможности управления не только расположением текстур на объекте (mapping), но и анимации этих текстур для достижения реалистичности, например, при изображении воды.

Вторая версия VRML стала намного сложнее первой. Если сначала можно было программировать движение фигур в простом текстовом редакторе (подобно первым алгоритмам двумерной компьютерной графики), на второй стадии потребовались специальные редакторы для создания vrml-объектов в vrml-файлах.

В 1998 г. незначительно переработанная спецификация VRML 2.0 под названием VRML 97 была принята в качестве официального стандарта ISO/IEC 1477.

Появление VRML 2.0 вызвало всплеск интереса к 3D в Сети как у пользователей, так и у разработчиков. В течение нескольких месяцев после принятия спецификации было разработано большое количество программ просмотра (браузеров) VRML 2.0.

Можно конвертировать трехмерные объекты из 3D Studio Max или AutoCAD.

Компания SGI возобновила свой интерес к VRML, купив ParaGraph International и выпустив программные продукты Cosmo Player и Cosmo World Builder. Однако менее чем через год фирма перестала поддерживать этот стандарт. Когда SGI сложила с себя бразды правления, спецификация поплыла в неопределенном направлении.

Пытаясь омолодить спецификацию, VRML Consortium был переименован в Web3D Consortium. Затем Web3D Consortium объявил о слиянии с World Wide Web Consortium (W3C). Тем самым они надеялись интегрировать VRML с другими сетевыми стандартами, такими как XHTML, XML, SVG, DOM, и SMIL.

Очередной удар VRML получил в начале 1999 года, когда фирма Platinum Technologies — один из ранних разработчиков VRML браузеров, отказалась от поддержки, в качестве последнего жеста доброй воли, объявила об открытии своих технологий для open source community.

Имелась надежда, что превращение VRML в открытый проект (как это было в самом начале), возобновит интерес к VRML тысяч новых независимых разработчиков. Но прежде чем Platinum успела опубликовать свои коды, она была куплена компанией Computer Associates, и исходники так и не были открыты.

Чем можно объяснить, что этот стандарт перестал развиваться? Основной причиной являлось преждевременность его появления.

В настоящее время появляется много новых 3D технологий. Некоторые из этих новых технологий лучше, быстрее, чем их vrml-предшественники. Некоторые из новых продуктов проще в использовании, некоторые включают новые интересные возможности. Однако в основе большинства из них лежит VRML 2.0.

MPEG-4 Interactive Profile (ISO/IEC 14496) был основан на VRML (теперь на X3D) и X3D, по большей части, обратно-совместим с ним.

VRML также продолжает использоваться в качестве файлового формата для обмена 3D-моделями, особенно в САПР.

Хотя VRML ещё продолжает использоваться в некоторых областях, особенно в образовательной и исследовательской сфере, где наиболее ценятся открытые спецификации, можно сказать, что он вытеснен форматом X3D

X3D — это стандарт ISO, предназначенный для работы с трёхмерной графикой в реальном времени. X3D — это наследник VRML. X3D является расширением VRML. В X3D возможно кодировать сцену используя синтаксис XML, равно как и Open Inventor - подобный синтаксис VRML97, а также расширенный интерфейс прикладного программирования (API).

VRML файлы имеют расширения **wrl** (от слова world - "мир") или **wrz**. В обоих случаях файл может быть либо текстовым (содержащим непосредственно код), либо gzip-архивом.

## *Единицы измерения*

В VRML приняты следующие единицы измерения:


- Расстояние и размер: метры
- Углы: радианы
- Остальные значения: выражаются, как часть от 1.
- Координаты берутся в трехмерной декартовой системе координат

## *Заголовок VRML-файла*

Как уже говорилось, vrml-документ представляет собой обычный тестовый файл. Для того, чтобы VRML-браузер распознал файл с VRML-кодом, в начале файла ставится специальный заголовок - file header:

```
#VRML V1.0 ascii
```

Такой заголовок обязательно должен находиться в первой строке файла, кроме того, перед знаком диеза не должно быть пробелов.

## *Примитивы VRML*

В VRML определены четыре базовые фигуры: куб (верней не куб, а прямоугольный параллелипипед), сфера, цилиндр и конус.

Эти фигуры называются примитивами (primitives). Набор примитивов невелик, однако комбинируя их, можно строить достаточно сложные трехмерные изображения. Например, вот такие:

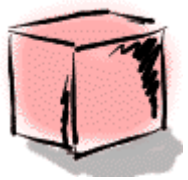
auto.wrl

Рассмотрим поподробней каждый из примитивов.

## *Куб*

Возможные параметры: width - ширина, height - высота, depth - глубина.

```
Cube {
      width   2    # ширина
      height  3    # высота
      depth   1    # глубина
     }
```



1cube.wrl

## *Сфера*

Параметр у сферы только один, это radius.

```
Sphere {
          radius  1    # радиус
        }
```

ball.wrl

## *Конус*

Возможные параметры: bottomRadius - радиус основания, height - высота, parts - определяет, какие части конуса будут видны. Параметр parts может принимать значения ALL, SIDES или BOTTOM.

```
Cone {
    parts        ALL  #видны и основание, и боковая поверхность конуса
    bottomRadius  1       #радиус основания
    height        2       #высота
   }
```

1cone.wrl

## *Цилиндр*

Для цилиндра можно задать параметры radius и height. Кроме того, с помощью параметра parts для цилиндра можно определить будут ли отображаться основания цилиндра и его боковая поверхность. Параметр parts может принимать значения ALL, SIDES, BOTTOM или TOP.

```
Cylinder {
    parts   ALL    #видны все части цилиндра
    radius  1      #радиус основания
    height  2      #высота цилиндра
        }
```

cylinder.wrl

## *Цвет и текстура*

Цвет фигуры, определяется с помощью объекта Material.

```
Material {
        ambientColor    0.2 0.2 0.2
        diffuseColor    0.8 0.8 0.8
        specularColor   0 0 0
        emissiveColor   0 0 0
        transparency    0
           }
```

Параметры ambientColor, diffuseColor, specularColor и emissiveColor управляют цветами и указываются в палитре RGB (красный, зеленый и голубой), причем первая цифра определяет интенсивность красного цвета, вторая - зеленого, а третья - синего.

К примеру, синий кубик, может быть описан следующим образом:

```
#VRML V1.0 ascii
     Material {
                diffuseColor 0 0 1
              }
      Cube { }
```

bluecube.wrl

Параметр transparency может принимать значения от 0 до1 и определяет степень прозрачности, причем максимальная прозрачность достигается при transparency равном единице. В следующем примере описано два цилиндра разных размеров, меньший из которых просвечивает сквозь другой.

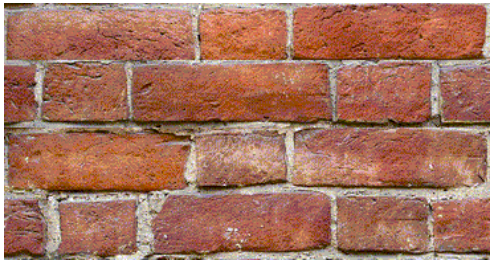
```
#VRML V1.0 ascii
    Material {
            diffuseColor 0 0 1
             transparency    0.7
        }
    Cylinder {
             height  1
              radius  1
        }
    Material {
              emissiveColor    1 0 0
              transparency    0
        }
    Cylinder {
                height  0.8
                radius  0.1
        }
```

2cil.wrl

Для имитирования различных поверхностей в VRML существует объект Texture2. В качестве текстуры легче всего использовать обычный графический файл, например, в GIF-формате. В таком случае для "натягивания" текстуры на трехмерное изображение нужно только указать путь к файлу в параметре filename объекта Texture2.

```
#VRML V1.0 ascii
  Texture2 {
            filename    "krp.gif"
            image       0 0 0
            wrapS REPEAT
            wrapT REPEAT
        }
  Cube {
            width 1
            height 1
            depth 1
      }
```

cubekrp.wrl



krp.gif

## *Положение объектов в пространстве*

### Изменение координат

По умолчанию любой описанный нами объект будет располагаться точно по центру окна браузера.

По этой причине, если мы опишем к примеру два одинаковых цилиндра, они сольются друг с другом. Для того, чтобы изменить положение второго цилиндра, применим узел Translation.

Узел Translation определяет координаты объекта:

```
Translation {
        translation 1 2 3        # т.е. соответственно x=1 y=2 z=3
            }
```

Вообще говоря, координаты указываемые в Translation не являются абсолютными. Фактически это координаты относительно предыдущего узла Translation. Чтобы прояснить это вопрос, рассмотрим пример:

```
#VRML V1.0 ascii

    Cube {
            width 1
            height 1
            depth 1
        }
    # Этот куб по умолчанию
располагается в центре

Translation {
            translation 2 0 0
              }
```

```
#Второй куб сдвинут вправо на 2
    Cube {
            width 1
            height 1
            depth 1
        }
    Translation {
            translation 2 0 0
            }
    #Третий куб сдвинут вправо на два
относительно 2-го !!!!
    Cube {
            width 1
            height 1
            depth 1
        }
```

Как видите, третий кубик вовсе не совпадает с первым, хотя в узле Translation указаны те же координаты.

В VRML 1.0 принято следующее правило: узлы, модифицирующие свойства фигур (Translation, Material и т.п.), действуют на все далее описанные фигуры.

Чтобы ограничить область действия модифицирующих узлов, фигуры необходимо сгруппировать с помощью узла Separator.

```
Separator
        {
          другие узлы
        }
```

Узел Separator работает как контейнер, он может содержать любые другие узлы, и основным его предназначением является именно ограничение области действия узлов типа Translation и Material.

Сравните следующий пример с предыдущим:

```
#VRML V1.0 ascii
      Separator {
        Cube {
            width 1
            height 1
            depth 1
             }
              }# конец области
действия узла Separator

Separator {
       Translation {
              translation 2 0 0
                 }
```

```
 #Второй куб сдвинут вправо на 2
        Cube {
            width 1
            height 1
            depth 1
            }
                }# конец области действия
узла Separator
       Separator {
        Translation {
                translation 2 0 0
                  }
            #Третий куб сдвинут вправо на
два относительно 1-го.
        Cube {
            width 1
            height 1
            depth 1
            }
                }# конец области действия
узла Separator
```

Хотя в примере описано три кубика, мы видим только два, так как второй и третий совпадают.

Рекомендуется всегда и везде использовать узел Separator. Он не только избавит от ошибок, связанных с относительностью координат, но и сделает VRML-код более простым и понятным.

## *Вращение*

Для вращения фигур вокруг осей координат применяется узел Rotation. (Углы в радианах)

```
Rotation {
        rotation 0 1 0  1.57
          }
```

Составим букву Т из двух цилиндров. По умолчанию цилиндр ориентирован вертикально. Поэтому для успешного выполнения задачи повернем его вокруг оси z на 90 градусов.

```
#VRML V1.0 ascii
   Separator { #Красный цилиндр
            Material {
                 diffuseColor 1 0 0
                 emissiveColor   1 0.6 0.6 }
            Cylinder {
                  height    1
                  radius    0.3
                   }
          }
```

t-cylind.wrl

```
 Separator { # Синий цилиндр,
повернутый на 90 градусов вокруг оси z
            Translation {
                 translation 0 0.5 0
                     }
            Rotation {
                  rotation    0 0 1  1.57
                   }
            Material {
                  diffuseColor 0 0 1
                  emissiveColor   0.5 0.5 1
                      }
            Cylinder {
                  height    1
                  radius    0.3
                   }
 }
```

## *Масштабирование*

Узел Scale масштабирует фигуры по одному или нескольким измерениям. Три цифры, стоящие после параметра scaleFactor определяют коэффициенты масштабирования относительно осей x, y и z.

```
Scale {
        scaleFactor 1 1 1
      }
```

В следующем примере, узел Scale сжимает сферу по оси x, и из сферы получается эллипсоид.

```
#VRML V1.0 ascii
 Material { emissiveColor   1 1 0 }
 Scale {
      scaleFactor 0.7 1 1 #сжимаем сферу по оси x
      }
 Sphere { radius 1}
```

ellips.wrl

## *Определение собственных объектов*

VRML предоставляет прекрасную возможность сократить и сделать более понятным исходный код VRML-файла путем описания собственных объектов. Это значит, что если в изображении несколько раз повторяется одна и та же фигура, то ее можно описать всего лишь один раз и в дальнейшем только ссылаться на нее.

Объект описывается одним из способов:

```
DEF name
    Cube { }
```

```
DEF name
    Material { }
```

```
DEF name
    Separator {
        Сгруппированные узлы,
        описывающие фигуру и
        свойства материала
    }
```

Для того, чтобы вставить в VRML-файл ранее определенную фигуру, используется команда USE

```
Separator {
        USE name
    }
```

Создадим VRML-файл, описывающий стул, при этом ножку стула опишем как объект LEG:

```
#VRML V1.0 ascii
   Material { emissiveColor 1 0.5 0.5 }
   Separator {
       Translation { translation 1 1 1 }
       DEF LEG  #Определяем объект - ножку стула
       Separator { # leg
          Cylinder {
             height  0.8
             radius  0.1
                  }
               } # определили ножку
            }
 Separator {
       Translation { translation 0 1 1 }
       USE LEG # используем определенный объект
          }
   Separator { # еще одна ножка
       Translation { translation 1 1 0 }
       USE LEG
          }
  Separator { # последняя ножка
       Translation { translation 0 1 0 }
       USE LEG
          }
```

stul.wrl

```
Separator { # сиденье
       Translation { translation 0.49 1.5 0.5 }
       Cube {
          height 0.2
          width 1.2
                        depth 1.2
           }
           }
   Separator { # спинка
      Translation { translation 0.49 2 0 }
      Cube {
        height 0.8
        width 1.2
        depth 0.2
         }
          }
   Separator { # закругление спинки
      Translation { translation 0.49 2.1 0 }
      Rotation {
        rotation    1 0 0 1.57
            }
      Cylinder {
        radius 0.6
        height 0.2
            }
     }
```

Как видите, нам не понадобилось описывать каждую ножку в отдельности - в результате объем VRML-кода стал меньше, а сам код более читабельным.

Еще один способ уменьшить размер VRML-файла - вставлять фигуры из другого файла.

Это позволяет делать узел WWWInline:

```
#VRML V1.0 ascii
   Separator {
                WWWInline {
                name        "    "
                bboxSize    0 0 0
                bboxCenter  0 0 0
              }
          }
```

Параметр name - это путь к файлу, параметры bboxSize и bboxCenter не обязательны и показывают пользователю размеры и положение вставляемого объекта, пока объект подгружается.

***Обратите внимание*** на две особенности VRML, незнание которых сильно затруднит создание VRML-документов вручную.

Все описания узлов и параметров в VRML регистрозависимы. Если Вы используете буквы неправильного регистра - то VRML - браузер просто проигнорирует такое описание.

В VRML имеет огромное значение порядок описания узлов. Так к примеру, следующие два описания дают совершенно разный результат.

```
...
    Rotation {...}
    Scale {...}
    ...
```

```
...
    Scale {...}
    Rotation {...}
    ...
```

# VRML97

Заголовок файла в VRML97

#VRML V2.0 utf8

Box (параллелепипед)

Box {size 2 2 2}

Text (текст)

```
Text { string []
      fontStyle NULL
       length []
       maxExtent 0.0
}
```

```
#VRML V2.0 utf8
Shape { geometry Text {string
["Hello,","world !"]} }
```

tut3_01.wrl

# PointSet (набор точек)

```
#VRML V2.0 utf8

Shape {
geometry PointSet {
coord Coordinate { point [ 0 0 0, 1 0 0, 2 0 0, 3 0 0, 4 0 0, 5 0 0,
0 1 0, 1 1 0, 2 1 0, 3 1 0, 4 1 0, 5 1 0,
0 2 0, 1 2 0, 2 2 0, 3 2 0, 4 2 0, 5 2 0,
0 3 0, 1 3 0, 2 3 0, 3 3 0, 4 3 0, 5 3 0,
0 4 0, 1 4 0, 2 4 0, 3 4 0, 4 4 0, 5 4 0,
0 5 0, 1 5 0, 2 5 0, 3 5 0, 4 5 0, 5 5 0
] }
color Color { color [ 1 0 0, 0 1 0, 0 0 1, 1 0 0, 0 1 0, 0 0 1,
0 1 0, 0 0 1, 1 0 0, 0 1 0, 0 0 1, 1 0 0,
0 0 1, 1 0 0, 0 1 0, 0 0 1, 1 0 0, 0 1 0,
1 0 0, 0 1 0, 0 0 1, 1 0 0, 0 1 0, 0 0 1,
0 1 0, 0 0 1, 1 0 0, 0 1 0, 0 0 1, 1 0 0,
0 0 1, 1 0 0, 0 1 0, 0 0 1, 1 0 0, 0 1 0
] }
}
}
```

## IndexedLineSet (линии по набору точек)

```
IndexedLineSet {
        color NULL
        coord NULL
        colorIndex [ ]
        colorPerVertex TRUE
        coordIndex [ ]
}
```

## IndexedFaceSet (грани по набору точек)

```
IndexedFaceSet {
        coord NULL
        color NULL
        normal NULL
        texCoord NULL
        ccw TRUE
        colorIndex [ ]
        colorPerVertex TRUE
        convex TRUE
        coordIndex
        creaseAngle 0
        normalIndex []
        normalPerVertex TRUE
        solid TRUE
        texCoordIndex [ ]
}
```

Это узел, которым можно заменить все остальные узлы, связанные с описанием граней. При экспорте в VRML код из какой-нибудь программы моделирующей 3D, получается файл, состоящий только из IndexedFaceSet. Принцип работы узла очень похож на IndexedLineSet: описан набор координат точек (coord) и указано, какие из них должны образовать грань (coordIndex).

Должны выполняться три условия:

- каждая грань должна состоять как минимум из трех несовпадающих вершин
- вершины должны задавать ПЛОСКИЙ многоугольник

многоугольник должен быть несамопересекающимся

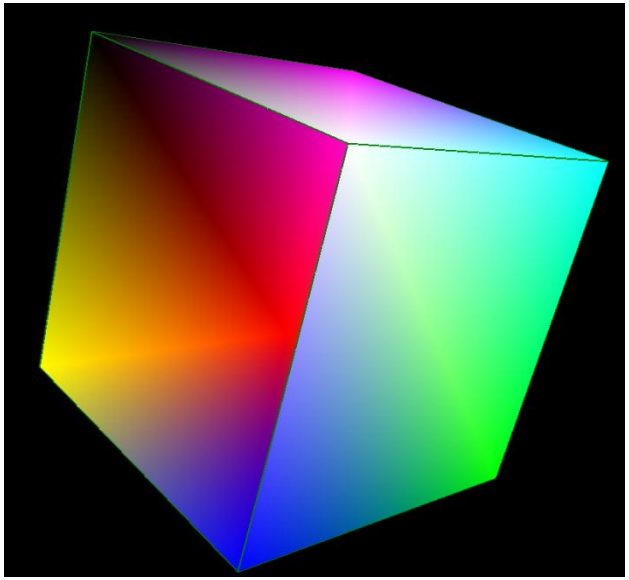
Легко догадаться, что все условия автоматически выполняются для треугольника, хотя в частном случае можете задавать плоские многоугольники с любым числом вершин.

Раскраска объектов в этом узле происходит так же, как и в IndexedLineSet:

- при colorPerVertex TRUE цвет приписывается ВЕРШИНАМ, а грань заливается градиентом между всеми вершинами, которыми грань создана.
- при colorPerVertex FALSE цвет приписывается каждой ГРАНИ в порядке, соответствующем порядку цветов в разделе color

Создадим для примера кубик без одной грани средствами узла IndexedFaceSet и раскрасим, пользуясь colorPerVertex TRUE

```
#VRML V2.0 utf8
Shape {geometry IndexedFaceSet {
        coord Coordinate {point [ -1 -1 -1, 1 -1 -1, 1 -1 1, -1 -1 1, -1 1 -1, 1 1 -1, 1 1 1, -1 1 1]}
        color Color {color [1 0 0, 0 1 0, 0 0 1, 1 1 0, 1 0 1, 0 1 1, 1 1 1, 0 0 0]}
        coordIndex [0 1 2 3 -1 4 5 6 7 -1 0 1 5 4 -1 0 3 7 4 -1 1 2 6 5 -1 ]
        solid FALSE
}                                       }
```



cube2.wrl

Введены операторы для примитивной анимации: сенсоры, маршруты, интерполяторы

Срабатывание сенсора может быть вызвано разными причинами: наступление определенного времени, клик мышкой, наведение курсора, приближение к объекту, столкновение с объектом и т.д.)

Интерполяторы выдают объекту численное значение какого-либо его параметра (цвет, положение, размер и т.д.) в данный момент времени в течение **cycleInterval**. За каждый **cycleInterval** интерполятор пробегает все значения полей **key** и **keyValue**.

Механизм route можно уподобить проводам, по которым передаются сигналы от **eventOut** к **eventIn** узлов или скриптов, можно от одного **eventOut** рассылать сообщения о событиях нескольким **eventIn**.

# Литература:

1. Марина Миланина, Diamond Team "VRML в примерах".