**А.А.Никитин, А.В.Никитин, Н.Н.Решетникова**

# ПРОГРАММИРОВАНИЕ НА VRML – ЯЗЫКЕ МОДЕЛИРОВАНИЯ ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ

**Учебное пособие**

**Санкт-Петербург**
**2001**



## Тема 1

## Простые геометрические формы, трансформация и внешний вид

**1. Узел Shape (Форма)**
Простые объекты добавляются к сцене с помощью узла **Shape**.
Синтаксис узла:

```
Shape {
        exposedField    SFNode          appearance      NULL
# представление геометрической формы опаределяет узел appearance
        exposedField    SFNode          geometry                NULL
# саму геометрическую форму определяет одиночный узел с именем geometry
}
```

**Пример (файл 1-0.wrl):** создаётся куб с размерами и цветом/прозрачностью по умолчанию:

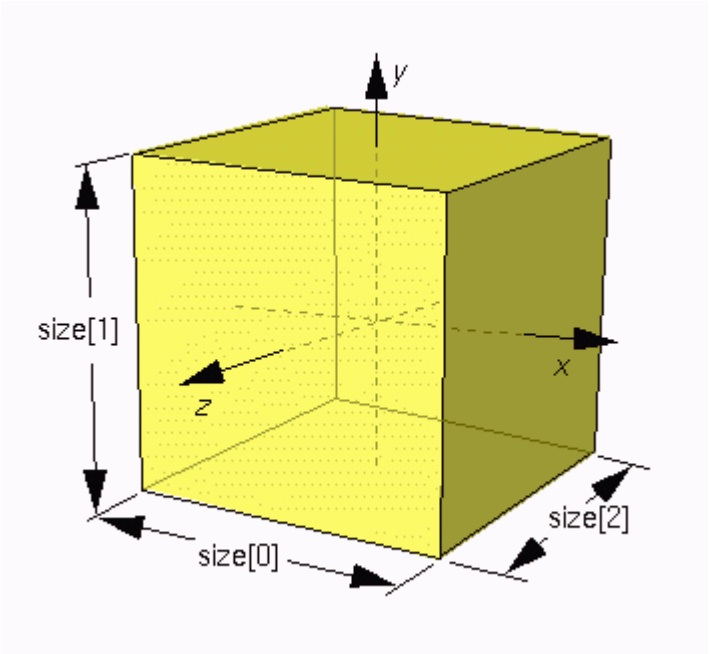
```
#VRML V2.0 utf8

Shape {
        appearance Appearance {
                material Material {
                }
        }
        geometry Box {
        }
}
```

В узле Shape необходимо использовать узлы appearance и geometry, которые описывают соответственно представление и форму объекта (см. далее).

**2.** В качестве значения поля geometry можно использовать один из следующих узлов, описывающих базовые примитивы языка VRML (куб, цилиндр, конус, сфера, текст).
Все примитивы изначально располагаются в координатах (0,0,0).

**2.1. Box (Куб )**
Синтаксис узла:

```
Box { field              SFVec3f         size                    2 2 2 }
```



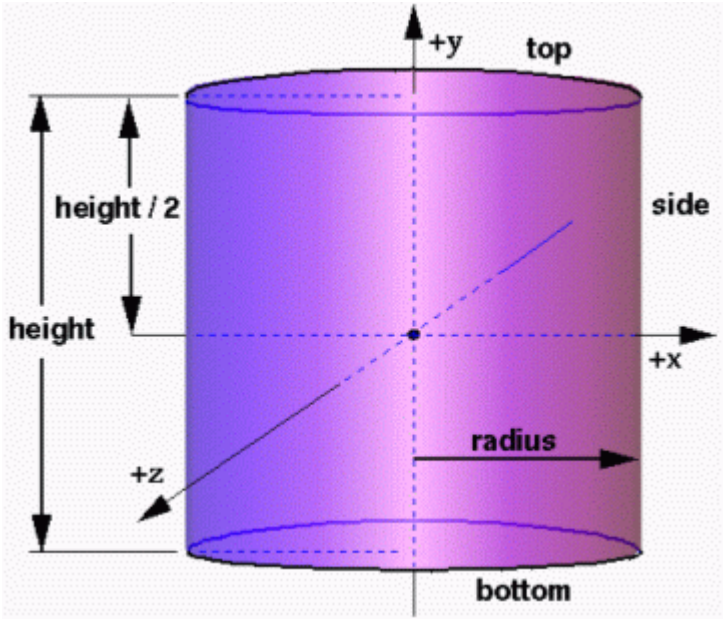

**Пример (файл 1-1.wrl):** создаётся параллелепипед 5.5x3.75x1.0 метров.

```
#VRML V2.0 utf8

Shape {
      appearance Appearance {
            material Material {
            }
      }
      geometry Box {
            size 5.5 3.75 1.0
      }
}
```

**2.2. Cylinder (Цилиндр)**
Синтаксис узла:

```
Cylinder {
        field           SFBool          bottom                  TRUE
        field           SFFloat         height                  2
        field           SFFloat         radius                  1
        field           SFBool          side                    TRUE
        field           SFBool          top                     TRUE
}
```

По умолчанию расположен в 0-координатах и имеет значения от +1 до –1 во всех измерениях.




Для определения радиуса и высоты цилиндра служат параметры radius и height соответственно. В этом узле существует ещё 3 параметра: side, top, и bottom, имеющие значения TRUE или FALSE (по умолчанию TRUE), которые сообщают браузеру, выводить ли соответственно верх, низ или сторону цилиндра (например, если у нас нижняя часть цилиндра перекрывается другим объектом, для облегчения работы (увеличения скорости работы) браузера можно выставить параметр bottom как FALSE).

**Пример (файл 1-2.wrl):** создаётся ”ведро” с радиусом основания 2.5 м и высотой 10м.

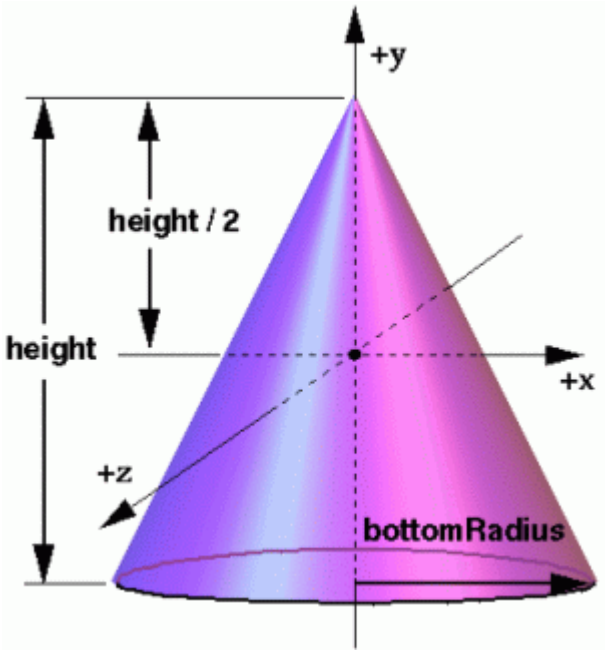
```
#VRML V2.0 utf8

Shape {
        appearance Appearance {
                material Material {
                }
        }
        geometry Cylinder {
                radius 2.5      #радиус (по умолчанию 1)
                height 10       #высота (по умолчанию 2)
                top FALSE       #цилиндр без верхней части
        }
}
```

### 2.3. Cone (Конус)

Синтаксис узла:

```
Cone {
        field           SFBool          bottom                  TRUE
        field           SFFloat         bottomRadius            1
        field           SFFloat         height                  2
        field           SFBool          side                    TRUE
}
```

По умолчанию конус будет на 1 метр выше и ниже оси Y, и радиусом основания 1 метр.



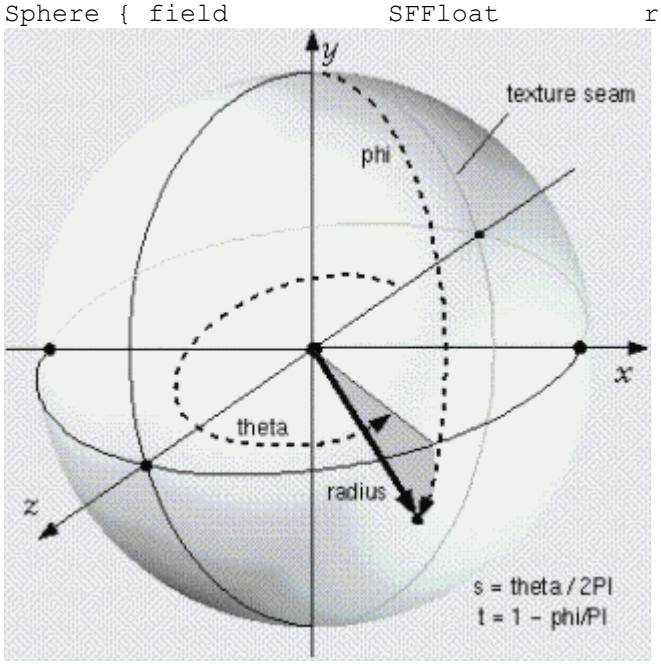

**Пример (файл 1-3.wrl):** создаётся ”колпак” с радиусом основания 5 м и высотой 10м.

```
#VRML V2.0 utf8

Shape {
      appearance Appearance {
            material Material {
            }
      }
      geometry Cone {
            bottomRadius 5    #радиус
            height 10         #высота
            side TRUE         #выводить ли сторону – выводить (по умолчанию TRUE)
            bottom FALSE      #выводить ли низ – не выводить (по умолчанию TRUE)
      }
}
```

### 2.4. Sphere (Сфера)

Синтаксис узла:

```
Sphere { field           SFFloat          radius                  1 }
```



**Пример (файл 1-4.wrl):** создаётся сфера радиусом 3 м.

```
#VRML V2.0 utf8

Shape {
        appearance Appearance {
                material Material {
                }
        }
        geometry Sphere {
                radius 3 #радиус сферы
        }

}
```

### 2.5. Text(Текст) и FontStyle(Стиль текста)

Синтаксис узлов:

```
Text {
        exposedField    MFString        string                  []
        exposedField    SFNode          fontStyle               NULL
        exposedField    MFFloat         length                  []
        exposedField    SFFloat         maxExtent               0.0
}

FontStyle {
        field           MFString        family                  "SERIF"
        field           SFBool          horizontal              TRUE
        field           MFString        justify                 "BEGIN"
        field           SFString        language                ""
        field           SFBool          leftToRight             TRUE
        field           SFFloat         size                    1.0
        field           SFFloat         spacing                 1.0
        field           SFString        style                   "PLAIN"
        field           SFBool          topToBottom             TRUE
}
```

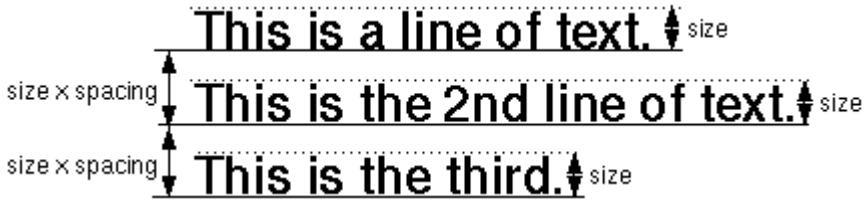
Узел FontStyle используется в качестве значения поля fontStyle узла Text.
С помощью узла Text в сцене создаётся 2D-текст.

Описание полей узла text:

- string – здесь перечисляются строки через “,”.
- fontStyle – в качестве значения этого поля используется узел FontStyle, определяющий параметры текста.
- maxExtent - максимальная ширина текста.
- length - длина каждой из строк через “,”.

Описание полей узла FontStyle:

- size - высота строки текста.
- family - тип шрифта - “SERIF”, “SANS” или “TYPEWRITER”.
- style - стиль шрифта – “PLAIN”, “BOLD”, “ITALIC” или “BOLD ITALIC”.
- horizontal - горизонтальный текст – TRUE, вертикальный – FALSE.
- leftToRight - TRUE(слева направо) или FALSE(наоборот).
- topToBottom - TRUE(сверху вниз) или FALSE(наоборот).
- language - код языка (2 символа).
- justify - выравнивание текста – “BEGIN”, “MIDDLE” или “END”.
- spacing - расстояние между строками текста (1-normal, 2-double spacing, и т.д.).




**Пример (файл 1-5.wrl):** создаётся текст (2 строки).

```
#VRML V2.0 utf8

Shape {
      appearance Appearance {
            material Material {
            }
      }
      geometry Text {
            string ["Virtual", "World"]    #символ ","-перевод на новую строку
            fontStyle FontStyle {          #параметры текста
                  size        2            #высота строки текста
                  family      "SERIF"      #тип шрифта - "SERIF", "SANS" или
                                           #"TYPEWRITER"
                  style       "PLAIN"      #стиль шрифта - "PLAIN", "BOLD",
                                           #"ITALIC" или "BOLD ITALIC"
                  horizontal  TRUE         #горизонтальный текст - TRUE,
                                           #вертикальный - FALSE
                  leftToRight TRUE   #TRUE(слева направо) или FALSE(наоборот)
                  topToBottom TRUE   #TRUE(сверху вниз) или FALSE(наоборот)
                  language    ""     #код языка (2 символа)
                  justify     "MIDDLE" #выравнивание текста
                  spacing           1     #расстояние между строками текста
            }

            maxExtent 5       #max ширина текста
            length [5, 5]     #длина каждой из строк
      }
}
```

### 3. Трансформация объектов

Для вращения, масштабирования и сдвига объектов используется узел Transform.

**Узел Transform (Трансформация).**
Синтаксис узла:

```
Transform {
        eventIn         MFNode          addChildren
        eventIn         MFNode          removeChildren
        exposedField    SFVec3f         center                  0 0 0
        exposedField    MFNode          children                []
        exposedField    SFRotation      rotation                0 0 1 0
        exposedField    SFVec3f         scale                   1 1 1
        exposedField    SFRotation      scaleOrientation        0 0 1 0
        exposedField    SFVec3f         translation             0 0 0
        field           SFVec3f         bboxCenter              0 0 0
        field           SFVec3f         bboxSize                -1 -1 -1
}
```

Описание полей:

- center – определяет центр локальной системы координат относительно оригинальной (0,0,0), в которой будет происходить трансформация.
- children – здесь находятся узлы (например, примитивы), подлежащие трансформации.

- rotation – указывается вектор, соответствующий оси вращения (3 числа – X Y Z), далее угол в радианах.
- scale – расширение/сжатие по осям X Y Z (указываются масштабные коэффициенты).
- scaleOrientation – определяет вращение системы координат перед масштабированием.
- translation – перемещение по осям X Y Z.
- bboxCenter | - эти поля определяют соответственно центр и размеры замкнутой
- bboxSize | геометрической области (куб), в которой находятся узлы из children
- addChildren | - входные события, позволяющие соответственно добавить или
- removeChildren | удалить узел для поля children.

**Пример (файл 1-6.wrl):** создаётся несколько объектов (3 объекта), одна из групп (2 объекта) объектов масштабируется, вращается и смещается, с третьим объектом никаких действий не производится .

```
#VRML V2.0 utf8

Transform {
      translation -8 -8 -8       #перемещение X Y Z
      rotation 0 1 0 0.78  #X Y Z - вектор, соответствующий оси вращения, далее
                        #угол в радианах (45 градусов)
                        #здесь вращаем вокруг оси Y
      scale 2 1 2       #расширение/сжатие X Y Z
      children [   #объекты
            Shape {
                  appearance Appearance {
                        material Material {
                        }
                  }
            geometry Box {
                  size 1 1 1
                  }
            }
            Shape {
                  appearance Appearance {
                        material Material {
                        }
                  }
            geometry Cone {
                  bottomRadius 0.3 #радиус
                  height 2     #высота
                  side TRUE    #выводить ли сторону (по умолчанию TRUE)
                  bottom TRUE  #выводить ли низ (по умолчанию TRUE)
            }
            }

      ]
}
Shape {
      appearance Appearance {
            material Material {
            }
      }
      geometry Sphere {
            radius 3 #радиус сферы
      }
}
```

**4. Внешний вид**

Поле appearance используется в узле Shape для задания узла Appearance.

**Узел Appearance (Внешний вид).**

Синтаксис узла:

```
Appearance {
        exposedField    SFNode          material                NULL
        exposedField    SFNode          texture                 NULL
        exposedField    SFNode          textureTransform        NULL
}
```

Например:

```
Shape {
        appearance Appearance {
                material Material {
                }
        }
        geometry Box {
        }
}
```

В этом примере в узле Appearance содержится поле material, которое содержит узел Material

**4.1 Узел Material (материал)**

Синтаксис узла:

```
Material {
        exposedField    SFFloat         ambientIntensity                0.2
        exposedField    SFColor         diffuseColor            0.8 0.8 0.8
        exposedField    SFColor         emissiveColor           0 0 0
        exposedField    SFFloat         shininess                       0.2
        exposedField    SFColor         specularColor           0 0 0
        exposedField    SFFloat         transparency            0
}
```

Узел Material может содержать любое из следующих полей:

- diffuseColor – нормальный цвет объекта (R G B, каждые R,G или B в интервале от 0 до 1; например, серый – 0.5 0.5 0.5).
- specularColor – цвет освещённых участков объекта (R G B).
- emissiveColor – каким цветом освещается объект - объект не отбрасывает свет на окружающие объекты (R G B).
- ambientIntensity – интенсивность отражаемого объектом света (значение от 0 до 1).
- shininess – определяет, в какой степени объект отражает свет (значение от 0 до 1).
- transparency – определяет прозрачность объекта – некоторые браузеры не поддерживают частично-прозрачные объекты (значение от 0 до 1)

**Пример (файл 1-7.wrl):** можно наблюдать полупрозрачный зелёный параллелепипед на фоне серой(по умолчанию) сферы.

```
#VRML V2.0 utf8

Transform {
      translation -8 -8 -8
      rotation 0 1 0 0.78
      scale 2 1 2
      children [   #объекты
            Shape {
            appearance Appearance {
```

```
                  material Material {
                        diffuseColor 0 0.5 0
                        emissiveColor 0 0.8 0
                        transparency 0.5
                  }
            }
            geometry Box {
            }
      }
      ]
}
Shape {
      appearance Appearance {
            material Material {
            }
      }
      geometry Sphere {
            radius 3 #радиус сферы
      }
}
```

**4.2. Наложение текстур.**
Поле texture узла Appearance может содержать один из узлов – ImageTexture, MovieTexture, PixelTexture.

**4.2.1. ImageTexture (Текстура-изображение)**
Синтаксис узла:

```
ImageTexture {
        exposedField    MFString        url                     []
        field           SFBool          repeatS                 TRUE
        field           SFBool          repeatT                 TRUE
}
```

с помощью этого узла можно покрыть объект текстурой формата JPEG или PNG. Некоторые браузеры поддерживают также GIF-формат текстур, но это нестандартно.
Описание полей:

- url - URL-ссылка на файл с изображением.
- repeatS – повторение текстуры по горизонтали (TRUE или FALSE).
- repeatT – повторение текстуры по вертикали (TRUE или FALSE).

Последние 2 поля реально полезны, когда объединены с узлом TextureTransform (см. далее). Если используется текстура типа GrayScale, параметр diffuseColor узла Material отражает интенсивность(насыщенность) текстуры. Можно создать ряд эффектов, комбинируя узел Material и узел ImageTexture.

**Пример (файл 1-8.wrl):** иллюстрирует наложение текстуры на сферу.

```
#VRML V2.0 utf8
Shape {
      appearance Appearance {
            texture ImageTexture {
            url "me.jpg"            #URL-ссылка на файл с изображением
            repeatS FALSE           #по умолчанию
            repeatT FALSE           #по умолчанию
            }
      }
      geometry Sphere {
            radius 3 #радиус сферы
      }
}
```

### 4.2.2. MovieTexture (Движущаяся текстура)

Синтаксис узла:

```
MovieTexture {
        exposedField    SFBool          loop                    FALSE
        exposedField    SFFloat         speed                   1
        exposedField    SFTime          startTime               0
        exposedField    SFTime          stopTime                0
        exposedField    MFString        url                     []
        field           SFBool          repeatS                 TRUE
        field           SFBool          repeatT                 TRUE
        eventOut        SFTime          duration_changed
        eventOut        SFBool          isActive
}
```

Узел MovieTexture наносит на объект в качестве текстуры MPEG-клип таким же образом, как узел ImageTexture изображение.
В узле MovieTexture содержатся те же 3 поля, что и в уле ImageTexture, но существуют и другие:

- speed – значение 1 означает стандартную скорость, 2 – ускоренный показ, 0 – отображается только первый кадр.
- loop – значение TRUE или FALSE, определяющее, повотряется клип или нет
- startTime – значение в секундах от 1st Jan 1970, показывающее, когда начинать проигрывать клип.
- stopTime – значение в секундах от 1st Jan 1970, показывающее, когда заканчивать проигрывать клип.

**Пример (файл 1-9.wrl):** на грани параллелепипеда накладывается текстура.

```
#VRML V2.0 utf8

Shape {
      appearance Appearance {
            texture MovieTexture {
                  url "viking.avi"
                  repeatS FALSE
                  repeatT FALSE
                  speed 1
                  loop TRUE
            }
      }
      geometry Box {
            size 5.5 3.75 1.0
      }
}
```

### 4.2.3. PixelTexture

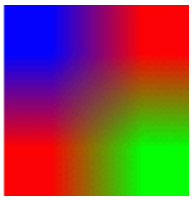
Синтаксис узла:

```
PixelTexture {
        exposedField    SFImage         image                   0 0 0
        field           SFBool          repeatS                 TRUE
        field           SFBool          repeatT                 TRUE
}
```

Этот узел позволяет определить свою текстуру в VRML(очень неэффективно).
В этом узле содержатся поля repeatS, repeatT и image (вместо URL).
Поле image содержит 2 числа, определяющие ширину и высоту текстуры, за ними следует другое число, определяющее число компонентов цвета.
1-компонентые цвета – в оттенках серого, 2-компонентные – в оттенках серого с прозрачностью, 3-к. – RGB-цвета, 4 к. - RGB-цвета с прозрачностью. После этих аргументов

следует список пикселей, представляющий собой набор чисел (2 разряда для каждой компоненты; например, 4-компонентный пиксель красного цвета и на 50% прозрачный будет определяться числом 0xFF00007F – в шестнадцатеричном формате). В списке пиксели упорядочиваются от левого нижнего угла к верхнему правому.

**Пример (файл 1-10.wrl):** куб и сфера раскрашиваются текстурой 2x2 пикселя вида



```
#VRML V2.0 utf8

Transform {
      translation -8 -8 -8       #перемещение X Y Z
      rotation 0 1 0 0.78  #X Y Z - вектор, соответствующий оси вращения, далее
                        #угол в радианах (45 градусов)
                        #здесь вращаем вокруг оси Y
      scale 2 1 2       #расширение/сжатие X Y Z
      children [   #объекты
            Shape {
            appearance Appearance {
                  texture PixelTexture
                  { image 2 2 3 0xFF0000 0x00FF00 0x0000FF 0xFF0000}
            }
            geometry Box {
            }
      }

      ]
}

Shape {
      appearance Appearance {
                  texture PixelTexture
                  { image 2 2 3 0xFF0000 0x00FF00 0x0000FF 0xFF0000}
      }
      geometry Sphere {
            radius 3 #радиус сферы
      }

}
```

### 4.3. Трансформация текстур

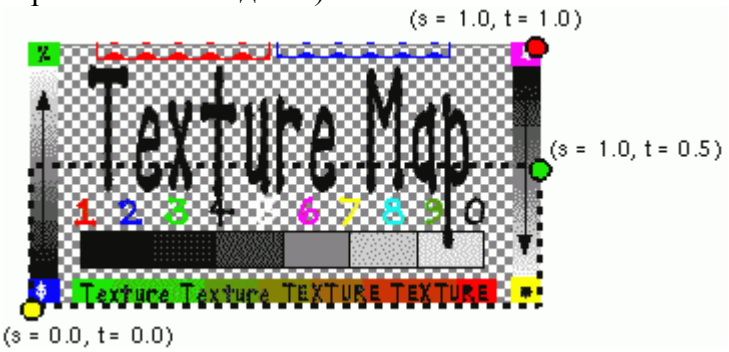
Способ, которым накладывается текстура, во-первых, зависит от самой геометрической фигуры. Во-вторых, можно у узла Appearance задать поле **textureTransform**. Значением этого поля должен быть узел TextureTransform, в котором описываются преобразования над текстурой. В-третьих, у узлов некоторых типов (IndexedFaceSet и ElevationGrid) настраивается поле textureCoordinate.

**Узел TextureTransform (Трансформация текстуры) –** при помощи этого узла можно определить, как размещать текстуру на объекте.
Структура узла:

```
TextureTransform {
      exposedField   SFVec2f   center          0 0
      exposedField   SFFloat   rotation        0
```

```
      exposedField   SFVec2f   scale            1 1
      exposedField   SFVec2f   translation      0 0
}
```

Текстуре присваивается локальная система координат ST.
S соответствует нижней части картинки, T - левой.
S и T изменяются от 0 (левый нижний пиксель) до 1 (правый нижний пиксель для S и левый верхний пиксель для T).




Значения параметров repeatS и repeatT определяют, будет ли текстура размножена в направлениях S и T, чтобы заполнить весь объект.

**Назначение полей:**

- center – задаётся точка в локальных координатах S и T, относительно которой происходит вращение текстуры.
- rotation – задаёт угол вращения текстуры в радианах.
- scale – масштабирует текстуру (задаются масштабные коэффициенты по S и T)
- translation – сдвиг текстуры по поверхности объекта.

**Пример (файл 1-11.wrl):**

```
#VRML V2.0 utf8
Shape {
      geometry Box {size 20 5 1 }
      appearance Appearance { material Material { diffuseColor 0 .3 .6}
            texture ImageTexture {
                  url ["stena.gif"]
                  repeatS FALSE
                  repeatT FALSE
            }
            textureTransform TextureTransform {
                  scale 10 2.5
                  center 0 0
                  rotation 0
                  translation 0 0.2
            }
      }
}
```

# Тема 2.

# ПРЕДСТАВЛЕНИЕ СЛОЖНЫХ ГЕОМЕТРИЧЕСКИХ ОБЪЕКТОВ

**Цель:** изучение представления сложных геометрических объектов на языке VRML с использованием узлов PointSet, IndexedLineSet, IndexedFaceSet, ElevationGrid, Extrusion и связанных с ними узлов Color, Coordinate, Normal, TextureCoordinate.

**Сложные геометрические объекты**
В качестве значения поля geometry(геометрия) узла Shape(форма) можно использовать один из следующих узлов, позволяющих создавать сложные геометрические объекты:

**1. PointSet (Набор точек)** – позволяет задать набор точек определённого цвета в пространстве.
Синтаксис узла:

```
PointSet {
        exposedField    SFNode          color                   NULL
#цвет
        exposedField    SFNode          coord                   NULL
#координаты
}
```

Рассмотрим ещё 2 узла, связанные с узлом PointSet – Color и Coordinate.

**Color (Цвет).**
Синтаксис узла:

```
Color {
        exposedField    MFColor         color                   []
}
```

С помощью этого узла можно описать значение цвета для каждой точки (последовательно задаётся цвет каждой точки по порядку).

**Coordinate (Коодинаты)**
Синтаксис узла:

```
Coordinate {
        exposedField    MFVec3f         point                   []
}
```

С помощью этого узла можно задать координату каждой точки (последовательно задаются координаты каждой точки по порядку).

Точки не участвуют в проверке на столкновения.

**Пример (файл 2-3.wrl) :** создаются 8 разноцветных точек в пространстве.

```
#VRML V2.0 utf8

Shape{
                appearance Appearance {
                # можно задать прозрачность точек и цвета
                # цвет каждой из точек можно определить непосредственно в самом
                # узле PointSet, как показано ниже
                # текстуры на точки наложить нельзя
        }
```

```
        geometry PointSet {
                color Color { #цвет каждой из 8 точек соответственно
                        color [1 0 0, 0 1 0, 0 0 1, 0 1 1,
                               1 0 1, 1 1 0, 1 1 1, 0.5 0.5 0]
                }

                coord Coordinate { # координата каждой из 8 точек
                          point [       -2 0 2, 2 0 2, 2 0 -2, -2 0 -2,
                                 -2 4 2, 2 4 2, 2 4 -2, -2 4 -2]
                }
        }
}
```

**2. IndexedLineSet (Индексированный набор линий) –** позволяет определить набор ломаных линий (которые могут образовывать многоугольники) определённого цвета в пространстве. Многоугольники не закрашены (состоят из линий).
Синтаксис узла:

```
IndexedLineSet {
        eventIn         MFInt32         set_colorIndex
        eventIn         MFInt32         set_coordIndex
        exposedField    SFNode          color                   NULL
        exposedField    SFNode          coord                   NULL
        field           MFInt32         colorIndex              []
        field           SFBool          colorPerVertex          TRUE
        field           MFInt32         coordIndex              []
}
```

Назначение полей:

- coord - определяет координаты вершин ломаных линий (так же, как в узле PointSet)
- coordIndex - определяет последовательность обхода координат из coord для построения ломаных линий. Нумерация координат в coord начинается с 0. Если описание одной из ломаных линий закончено, ставится маркер окончания ломаной линии (–1) - см. пример.
- colorPerVertex – если значение TRUE, то цвета, перечисленные в узле color, соответствуют вершинам ломаных линий. Иначе цвета соответствуют ломаным линиям целиком.
- color - определяет цвета ломаных линий или их сегментов (цвета записываются, как в PointSet).
- colorIndex – в этом поле перечисляется, какой цвет из узла Color какой вершине(или линии) соответствует. Перечисляются порядковые номера цветов (0:N-1).

Например:
А) пусть у нас в узле Color определено 3 цвета (№ 0-2).

```
colorPerVertex  FALSE
colorIndex      [1 2 0]  #вершине №0 соответствует цвет №1 из списка цветов
                         #вершине №1 соответствует цвет №2
                         #вершине №2 соответствует цвет №0
```

Б) пусть в узле Color 6 цветов (№ 0-5).

```
      coord Coordinate { #здесь задаются 8 координат
            point [ -2 0 2, 2 0 2, 2 0 -2, -2 0 -2
                    -2 4 2, 2 4 2, 2 4 -2, -2 4 -2]
      }
      # здесь задаётся последовательность обхода координат, заданных выше
      # (по номерам: первая координата №0, вторая №1, последняя №7)
      coordIndex [ 0 1 2 3 0 -1  #это координаты ломаной линии №0
                                 #первая ломаная линия является цепочкой координат
                                 #с номерами 0-1-2-3-0 – это замкнутый
                                 #многоугольник с
```

```
                                # четырьмя вершинами
                                #-1 показывает, что координаты первой ломаной
                                #линии
                                #закончились
                  2 6 5 1 -1    #координаты ломаной №1
                  ]

        colorPerVertex  TRUE    #по умолчанию
# далее индексов должно быть перечислено столько же, сколько в поле coordIndex,
#включая маркеры
# окончания ломаных линий
        colorIndex      [1 3 5 2 1 -1   #данные номера цветов соответствуют
                         0 4 3 5 -1       #номерам вершин из поля coordIndex
                        ]
```

Если индексы не указаны, то цвета соответствуют поверхностям (вершинам) в том порядке, в котором они перечислены (цвет № n соответствует вершине № n, т.е. цвет № 0 вершине № 0, цвет № 1 вершине № 1 и т.д.).

- set_colorIndex – входное событие, с помощью которого можно установить поле colorIndex.
- set_coordIndex - входное событие, с помощью которого можно установить поле coordIndex.

Линии не участвуют в проверке на столкновение.

**Пример (файл 2-2.wrl) :** рисуются 2 многоугольника, образующих 2 перпендикулярные плоскости. Значения цветов в вершинах линий интерполируются по поверхностям линий.

```
#VRML V2.0 utf8

Shape{
            appearance Appearance {
                  material Material {
                        #здесь можно определить цвета и прозрачность
                        #цвет можно также определить непосредственно в самом узле
                        #IndexedLineSet, как показано ниже
                        #текстуры на ломаные линии не накладываются
                  }
            }
geometry IndexedLineSet {
      color Color {
            color[0 0 1,0 1 0,0 1 1,1 0 0,1 0 1,1 1 0,1 1 1]
      }
      coord Coordinate { #здесь задаются 8 координат
            point [     -2 0 2, 2 0 2, 2 0 -2, -2 0 -2,
                  -2 4 2, 2 4 2, 2 4 -2, -2 4 -2]
      }
      # здесь задаётся последовательность обхода координат, заданных выше
      # (по номерам: первая координата №0, вторая №1, последняя №7)
      coordIndex [ 0 1 2 3 0 -1  #это координаты ломаной линии №0
                                 #первая ломаная линия является цепочкой координат
                                 #с номерами 0-1-2-3-0 – это замкнутый
                                 #многоугольник с
                                 # четырьмя вершинами
                                 #-1 показывает, что координаты первой ломаной
                                 #линии закончились
                  2 6 5 1 -1     #координаты ломаной №1
                 ]
      colorIndex[ 0 1 2 3 4 -1
                  5 6 0 1 -1
              ]
}
}
```

**3. IndexedFaceSet (Индексированный набор поверхностей) –** этот узел служит для создания сложных геометрических фигур (с внутренней структурой или без неё), состоящих из набора поверхностей.

Синтаксис узла:

```
IndexedFaceSet {
        eventIn         MFInt32         set_colorIndex
        eventIn         MFInt32         set_coordIndex
        eventIn         MFInt32         set_normalIndex
        eventIn         MFInt32         set_texCoordIndex
        exposedField    SFNode          color                   NULL
        exposedField    SFNode          coord                   NULL
        exposedField    SFNode          normal                  NULL
        exposedField    SFNode          texCoord                NULL
        field           SFBool          ccw                     TRUE
        field           MFInt32         colorIndex              []
        field           SFBool          colorPerVertex          TRUE
        field           SFBool          convex                  TRUE
        field           MFInt32         coordIndex              []
        field           SFFloat         creaseAngle             0
        field           MFInt32         normalIndex             []
        field           SFBool          normalPerVertex         TRUE
        field           SFBool          solid                   TRUE
        field           MFInt32         texCoordIndex           []
}
```

Перед описанием полей данного узла рассмотрим ещё один узел – Normal.
**Normal (Нормаль) –** определяет список векторов нормалей.
Синтаксис узла:

```
Normal {
        exposedField    MFVec3f         vector                  []
}
```

Например поле:

```
normal Normal {
        vector [1 0 1,
                -1 0.5 -1,
                -2 0 -3
        ]
}
```

определяет 3 вектора нормалей с соответствующими координатами.
Нормали нужны для вычисления того, как поверхность будет освещена. Если мы изменим значение нормали для поверхности, то в браузере мы увидим, как поменяется освещение этой поверхности.

Описание полей:

- color - так же, как в IndexedLineSet.
- coord - координаты вершин (узел Coordinate - как в IndexedLineSet).
- normal - список нормалей (в качестве значения применяется узел Normal).
- texCoord - применяется для нанесения текстуры (см. ниже узел TextureCoordinate).
- ccw - если TRUE, то видимые поверхности, перечислямые в coordIndex – это те, номера вершин у которых перечислены против часовой стрелки. Если ccw FALSE, то видимые поверхности это те, номера вершин которых перечислены по часовой стрелке.
- colorIndex - так же, как в IndexedLineSet (в зависимости от значения colorPerVertex).

- colorPerVertex - цвета соответствуют поверхностям (если значение FALSE) или вершинам (TRUE).
- convex - если в фигуре присутствуют невыпуклые (неплоские, самопересекающиеся) многоугольники, то значением данного поля следует указать FALSE (иначе результат работы браузера не определён).
- coordIndex - соответствие координат поверхностям – список номеров координат с маркером окончания (-1) для каждой поверхности. Номера вершин для видимой поверхности перечисляются против часовой стрелки (если ccw TRUE) если значение solid=true. Если значение solid=false (ccw TRUE), то поверхность видна с двух сторон.
- creaseAngle - если поле normal пусто (NULL), то браузер использует этот угол для вычисления того, как поверхность будет освещаться. В зависимости от значения этого поля у поверхности будет гладкое или негладкое затенение. Значение этого поля – угол, который определяет, как генерируются нормали по умолчанию. Если угол между геометрическими нормалями двух смежных поверхностей меньше, чем creaseAngle, то нормали вычисляются таким образом, что поверхности по краю имеют гладкое затенение.
- normalIndex – перечисляются номера нормалей на соответствие вершинам или поверхностям (в зависимости от значения normalPerVertex) – по такому же принципу, как в colorIndex.
- normalPerVertex - нормали соответствуют поверхностям (FALSE) или вершинам (TRUE).
- solid - см. coordIndex.
- texCoordIndex - значением этого поля является узел TextureCoordinate (см. ниже).

**Пример (файл 2-1.wrl) :** определяется 4 полупрозрачные плоскости (видимые с двух сторон), определяющие ½ куба. Цвета соответствуют вершинам (интерполируются между ними).

```
#VRML V2.0 utf8
Shape{
                appearance Appearance {
                        material Material {
                                transparency 0.5  # полупрозрачный материал
                        }
                }
geometry IndexedFaceSet {
        coord Coordinate {
                point [ -2 0 2, 2 0 2, 2 0 -2, -2 0 -2,
                        -2 4 2, 2 4 2, 2 4 -2, -2 4 -2]
        }
        coordIndex [    0 4 7 3 -1
                        1 2 6 5 -1
                        4 5 6 7 -1
                        2 3 7 6 -1 ]
        solid FALSE
        color Color {
            color[0 0 1,0 1 0,0 1 1,1 0 0,1 0 1,1 1 0,1 1 1]
        }
        colorIndex [    0 1 2 3 -1
                        4 5 6 0 -1
                        1 2 3 4 -1
                        5 6 0 1 -1 ]

}
}
```

**Пример (файл 2-1-1.wrl) :** определяется 4 плоскости, каждая видна только с одной стороны. Цвета соответствуют вершинам (интерполируются между ними).

```
#VRML V2.0 utf8
Shape{
                appearance Appearance {
                        material Material {
```

```
                        }
                }
geometry IndexedFaceSet {
        coord Coordinate {
                point [ -2 0 2, 2 0 2, 2 0 -2, -2 0 -2,
                        -2 4 2, 2 4 2, 2 4 -2, -2 4 -2]
        }
        coordIndex [    0 4 7 3 -1
                        5 6 2 1 -1
                        4 5 6 7 -1
                        6 7 3 2 -1 ]
        color Color {
            color[0 0 1,0 1 0,0 1 1,1 0 0,1 0 1,1 1 0,1 1 1]
        }
        colorIndex [    0 1 2 3 -1
                        4 5 6 0 -1
                        1 2 3 4 -1
                        5 6 0 1 -1 ]

}
}
```

**Пример (файл 2-1-2.wrl) :** это пример по нормалям. Для первой (слева направо) пирамиды нормали не определяется, у второй пирамиды нормали соответствуют поверхностям, у третьей – вершинам. Можно посмотреть, как меняется освещение на каждой из пирамид.

**4. ElevationGrid (Сетка возвышенностей) –** служит для создания поверхностей, определяя набор(сетку) возвышенностей над горизонтальной плоскостью (над Y=0).
Синтаксис узла:

```
ElevationGrid {
        eventIn         MFFloat         set_height
        exposedField    SFNode          color                   NULL
        exposedField    SFNode          normal                  NULL
        exposedField    SFNode          texCoord                NULL
        field           MFFloat         height                  []
        field           SFBool          ccw                     TRUE
        field           SFBool          colorPerVertex          TRUE
        field           SFFloat         creaseAngle             0
        field           SFBool          normalPerVertex         TRUE
        field           SFBool          solid                   TRUE
        field           SFInt32         xDimension              0
        field           SFFloat         xSpacing                0.0
        field           SFInt32         zDimension              0
        field           SFInt32         zSpacing                0.0
}
```

Назначение полей:

- color - |
- normal - |
- texCoord - |
- ccw - |--------> как в узле IndexedFaceSet
- colorPerVertex - |
- creaseAngle - |
- normalPerVertex- |
- solid - |

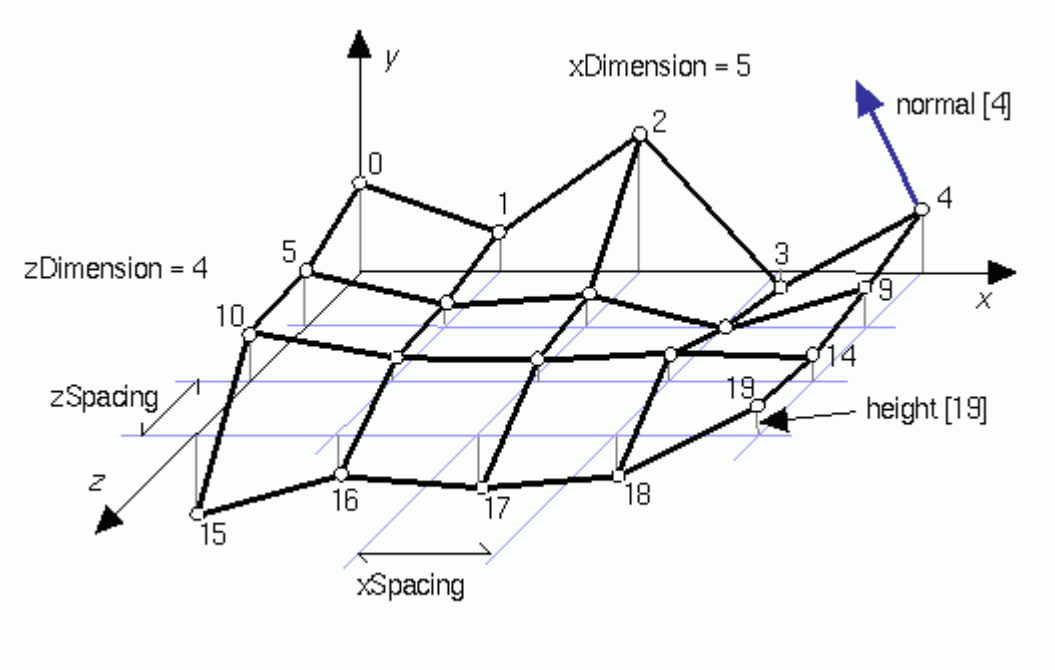
- xDimension - | - эти переменные определяют размерность сетки возвышенностей.
- zDimension - | Соответственно поле height должно содержать xDimension*zDimension

| компонент

- xSpacing - | - здесь определяется расстояние между соседними вершинами в метрах
- zSpacing - | соответственно по осям X и Z

- height - это поле определяет набор возвышенностей в метрах над горизонтальной плоскостью Y=0. Если смотреть на поверхность сверху в направлении оси –Z, то вершины перечисляются от левого верхнего угла к правому нижнему (построчно).

- set_height - входное событие, устанавливающее вершины (поле height) – это для создания движущихся поверхностей.




**Пример (файл 2-4.wrl) :** рисуется поверхность.

```
#VRML V2.0 utf8

Shape{
                appearance Appearance {
                        material Material {
                                diffuseColor 0 0.5 0
                                emissiveColor 0 0.8 0
                                transparency 0.5
                        }
                }
geometry ElevationGrid {
        xDimension 7
        zDimension 6
        height [1.5, 1, 0.5, 0.5, 1, 1.5,0,
                1, 0.5, 0.25, 0.25, 0.5, 1,0,
                0.5, 0.25, 0, 0, 0.25, 0.5,0,
                0.5, 0.25, 0, 0, 0.25, 0.5,0,
                1, 0.5, 0.25, 0.25, 0.5, 1,0,
                1.5, 1, 0.5, 0.5, 1, 1.5,0]
        xSpacing 5.0
        zSpacing 5.0
        solid FALSE
        }
}
```

**5. Extrusion (Вытеснение) –** позволяет строить сложные геометрические объекты.
Синтаксис узла:

```
Extrusion {
        eventIn         MFVec3f         set_crossSection
        eventIn         MFRotation      set_orientation
        eventIn         MFVec2f         set_scale
        eventIn         MFVec3f         set_spine
        field           SFBool          beginCap                TRUE
        field           SFBool          ccw                     TRUE
        field           SFBool          convex                  TRUE
        field           SFFloat         creaseAngle             0
        field           MFVec2f         crossSection            [1 1, 1 -1, -1 -1,
                                                                -1 1, 1 1]
        field           SFBool          endCap                  TRUE
        field           MFRotation      orientation             0 0 1 0
        field           MFVec2f         scale                   1 1
        field           SFBool          solid                   TRUE
        field           MFVec3f         spine                   [0 0 0, 0 1 0]
}
```

Принцип работы такой: описывается многоугольник в горизонтальной плоскости Y=0 и траектория его движение, а также масштаб этого многоугольника в различных точках траектории.
Назначение полей:

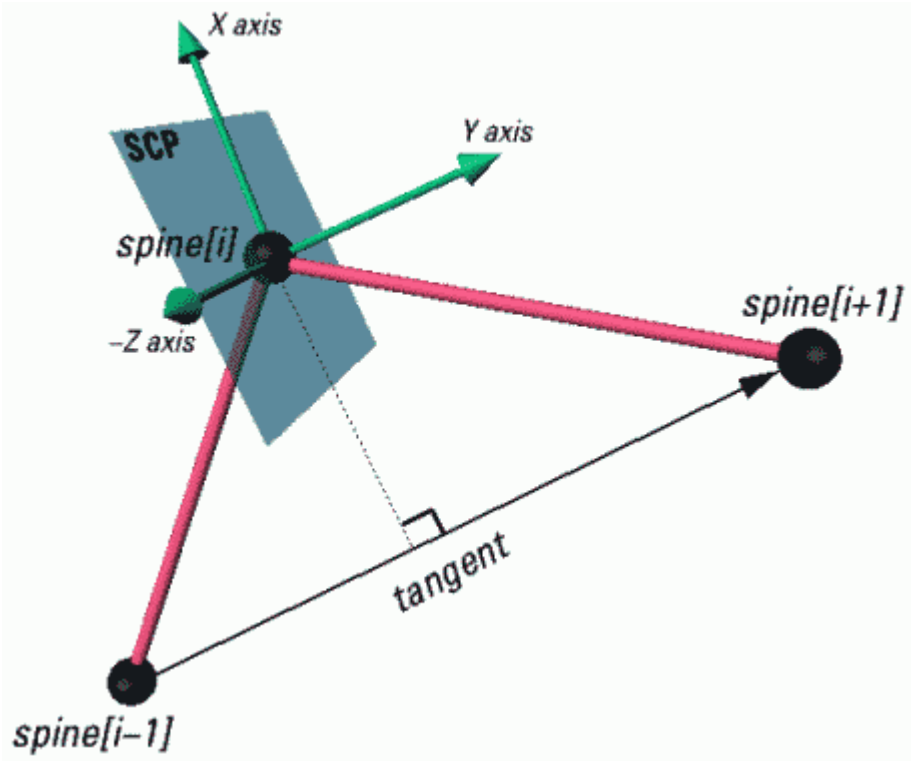
- set_crossSection- |
- set_orientation - |-входные события для установки соответственно полей crossSection,
- set_scale - | orientation, scale, spine
- set_spine - |

- beginCap - | эти поля определяют, будут ли закрыты торцы объекта
- endCap - | (т.е. если эти значения, а также solid (см. ниже) установить в FALSE, то можно будет видеть объект снаружи, а также зайти внутрь через торцы.

- ccw - при значении solid TRUE, если ccw TRUE, то будет видна сторона многоугольника, вершины которой перечислены против часовой стрелки (если смотреть на объект извне), если ccw FALSE, то видимая сторона та, вершины которой перечислены по часовой стрелке.

- convex - | - как у IndexedFaceSet.
- creaseAngle - |

- crossSection - описывает координаты X и Z многоугольника в горизонтальной плоскости Y=0.
- orientation - задаётся направление оси вращения объекта и угол в радианах.
- scale - масштабные коэффициенты многоугольника по X и Z в каждой точке траектории. Если значение (1 1), то исходное сечение не масштабируется, иначе число значений должно совпадать с числом точек в траектории.
- solid - будет видна одна сторона многоугольника (TRUE) или обе (FALSE) – см. также ccw.
- spine - описывает траекторию по точкам (X Y Z).




**Пример (файл 2-5.wrl) :** построение сложной поверхности с основанием в виде квадрата.

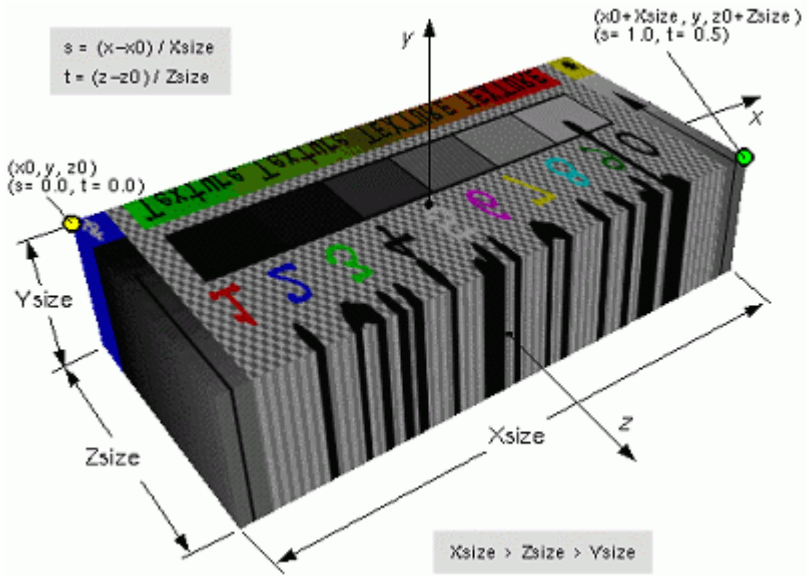
```
#VRML V2.0 utf8

Shape{
                appearance Appearance {
                        material Material {
                        }
                }
geometry Extrusion {
        solid FALSE
        beginCap FALSE
        endCap FALSE
        crossSection [1 1, 1 -1, -1 -1, -1 1, 1 1]
        spine [0 0 0, 0 2 0, 1 3 0, 2 3 0]
        scale [1 1, 1 0.5, 0.5 1, 0.5 0.5]
        }

}
```

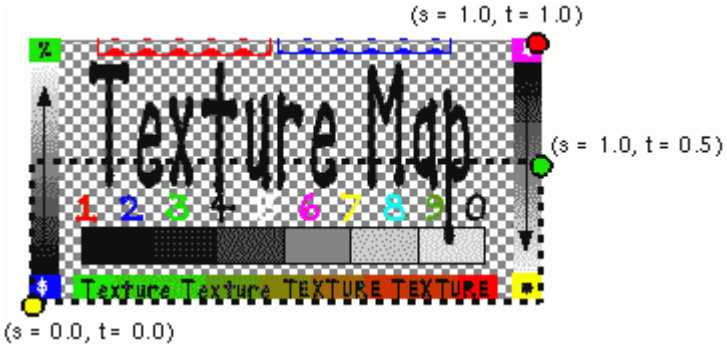
6. В узлах, где присутствует поле texCoord, в качестве его значения можно использовать узел TextureCoordinate.

**TextureCoordinate (Координаты текстуры)**
Синтаксис узла:

```
TextureCoordinate {
        exposedField    MFVec2f         point                   []
}
```

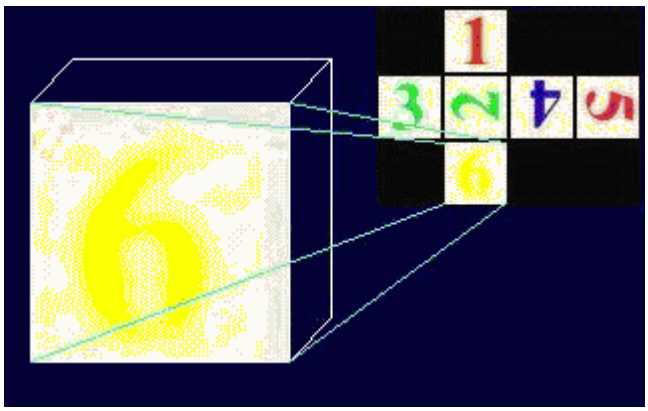
В поле point перечисляются относительные координаты текстуры s и t.
Вычисляются s и t исходя из следующих изображений :




Далее в поле texCoordIndex перечисляется соответствие координат текстуры (из узла TextureCoordinate) вершинам объекта. Для IndexedFaceSet поле texCoordIndex должно содержать столько же индексов (вместе с маркерами окончания поверхности), сколько и поле coordIndex, только вместо координат вершин должны быть указаны координаты текстур для соответствующих вершин.

**Пример (файл 2-6.wrl) :** для нанесения на куб, созданный при помощи узла IndexedFaceSet, текстуры вида



необходим следующий программный код:

```
#VRML V2.0 utf8

Shape {
      appearance Appearance {
            texture ImageTexture {
                  url "d6.jpg"
```

```
        }
    }
    geometry IndexedFaceSet {
        coord DEF COORD Coordinate {  # координаты вершин
            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]
        }
        coordIndex [ 3 2 1 0 -1 # описание 6 поверхностей (номера координат
                    0 1 5 4 -1  # вершин
                    1 2 6 5 -1  # с маркерами окончания описания поверхности)
                    2 3 7 6 -1
                    3 0 4 7 -1
                    4 5 6 7 -1
                    ]
        texCoord DEF TEXCOORD TextureCoordinate {
            point [ 0.25 0, 0.5 0, # относительные координаты вершин
                    0 0.33, 0.25 0.33, 0.5 0.33, 0.75 0.33, 1 0.33,
                    0 0.67, 0.25 0.67, 0.5 0.67, 0.75 0.67, 1 0.67,
                    0.25 1, 0.5 1 ]
        }
        texCoordIndex [5 6 11 10 -1   # координаты текстур, соответствующие
                       13 12 8 9 -1   #описанию поверхности
                       7 2 3 8 -1
                       0 1 4 3 -1
                       5 10 9 4 -1
                       9 8 3 4 -1
        ]
    }
}
```

# Тема 3.

# ГРУППИРОВАНИЕ, ГИПЕР-ССЫЛКИ И ТИРАЖИРОВАНИЕ

### 3.1. Группирование.

Группирующими узлами являются следующие:

- Anchor
- Billboard
- Collision
- Group
- LOD
- Switch
- Transform

Рассмотрим узел Group.

**Узел Group(Группа).**
Синтаксис узла:

```
Group {
        eventIn         MFNode          addChildren
        eventIn         MFNode          removeChildren
        exposedField    MFNode          children                []
        field           SFVec3f         bboxCenter              0 0 0
        field           SFVec3f         bboxSize                -1 -1 -1
}
```

Назначение полей:

- addChildren | - это входные события соответственно для добавления/удаления
- removeChildren | потомков
- children – потомки
- bboxCenter | - эти 2 поля определяют центр и размеры параллелепипеда, в пределах
- bboxSize | которого существуют дочерние узлы. Это может использоваться с целью | оптимизации сцены. Результат не определен, если в размеры параллелепипеда | не влезает хотя бы один из потомков. Значение по умолчанию этих двух узлов | подразумевает, что параллелепипед не существует.

Примеры:

Узлы записываются как пример не вложенных узлов:

```
Group { # пример невложенных узлов
        children [
                # здесь содержатся дочерние узлы
        ]
}
```

Узлы могут быть вложены друг в друга, создавая иерархию узлов, часто называемую графом сцены:

```
Group {         #пример вложенных узлов
        children [      #поле children может содержать другие узлы
                Group {
                }
        ]
}
```

С помощью узла Group и команды DEF(см. ниже) можно определит группу объектов, а также события eventIn и eventOut (см. след. часть) для этой группы.

**3.2.** Гипер-ссылку можно прикрепить к объекту (группе объектов), используя узел Anchor.

**Узел Anchor (Якорь) – используется для перехода на URL.**
Синтаксис узла:

```
Anchor {
        eventIn         MFNode          addChildren
        eventIn         MFNode          removeChildren
        exposedField    MFNode          children                []
        exposedField    SFString        description             ""
        exposedField    MFString        parameter               []
        exposedField    MFString        url                     []
        field           SFVec3f         bboxCenter              0 0 0
        field           SFVec3f         bboxSize                -1 -1 -1
}
```

Назначение полей:

- addChildren | - это входные события соответственно для добавления/удаления
- removeChildren | потомков.
- children – потомки (к которым прикрепляется гипер-ссылка).
- description - описание гипер-ссылки (текст, который отображается, когда курсор находится над объектам, к которому прикреплена гипер-ссылка).
- parameter - параметры перехода (например: parameter [“target=frame_1”] , т.е. в какой кадр загружать).
- url - URL, на который осуществляется переход.
- bboxCenter | - эти 2 поля определяют центр и размеры параллелепипеда, в пределах
- bboxSize | которого существуют дочерние узлы. Это может использоваться с целью | оптимизации сцены. Результат не определен, если в размеры параллелепипеда | не влезает хотя бы один из потомков. Значение по умолчанию этих двух узлов | подразумевает, что параллелепипед не существует.

|

Другими словами, они касаются обеспечения ограниченной сферы для VRML-браузера для увеличения скорости вычислений при представлении сцены. При их использовании нужно быть уверенным, что параметры указаны достаточно большими для отображения всех объектов, указанных в поле children.

**Пример (файл 3-1.wrl):** наращиваем файл 1-7.wrl – получаются 2 объекта (куб и сфера), с которых можно осуществить переходы на просмотр видео и сцены 1-1.wrl.

```
#VRML V2.0 utf8
Anchor
{
  children
  [
    Transform {
      translation -8 -8 -8       #перемещение X Y Z
      rotation 0 1 0 0.78  #X Y Z - вектор, соответствующий оси вращения, далее
                        #угол в радианах (45 градусов)
                        #здесь вращаем вокруг оси Y
      scale 2 1 2       #расширение/сжатие X Y Z
      children [   #объекты
            Shape {
            appearance Appearance {
                  material Material {
                        diffuseColor 0 0.5 0
                        emissiveColor 0 0.8 0
                        transparency 0.5
                  }
            }
            geometry Box {
            }
```

```
      }
      ]
    }
  ]
  description "Переход к просмотру видео"
  url "viking.avi"
}
Anchor
{
  children
  [
    Shape {
      appearance Appearance {
            material Material {
            }
      }
      geometry Sphere {
            radius 3 #радиус сферы
      }

    }
  ]
  description "Переход к сцене 1-1.wrl"
  url "1-1.wrl"
}
```

### 3.3. Тиражирование с использованием операторов DEF/USE

Если в сцене используются идентичные объекты, можно многократно использовать предыдущие определения объекта, например, так:

```
#определяем куб как объект с именем FBOX
DEF FBOX Shape {
        appearance Appearance {
                material Material {
                }
        }
        geometry Box {
        }
}
#когда мы хотим использовать куб опять, пользуемся ключевым словом USE
Transform
{       translation 10 10 10
        children [USE FBOX]
}
```

**Пример (файл 3-2.wrl):** создаем куб и тиражируем его.

```
#VRML V2.0 utf8
#создаём куб, расположенный в (0,0,0) размерами 1x1x1
DEF KUB1 Shape {
      appearance Appearance {
            material Material {
            }
      }
      geometry Box {
            size 1 1 1
      }
}

Transform
{
      translation 3 0 0
      children [USE KUB1]
}
```

### 3.4. Прототипы

Использование прототипов – один из путей использования одного и того же кода. Если необходимо использование однотипных объектов, можно пользоваться командами DEF и USE. Команда PROTO ипользуется при необходимости создания объекта, который очень похож на уже существующий, но имеет небольшие отличия (например, различные высоты).
Для определения прототипа на первом шаге необходимо определить объект, а также поля и события для него.

Пример использования команды PROTO для кубов с различными цветами:

```
PROTO VBox [
        field SFColor boxColour 1 0 0
]
{       Shape {
                appearance Appearance {
                        material Material {
                                diffuseColor IS boxColour
                        }
                }
                geometry Box {
                }
        }
}
```

Поле, заключённое в [] – интерфейс для объекта (числа – определённые значения по умолчанию для поля).
Когда образец VBox объявлен, то величина, приписанная полю boxColour помещается в поле diffuseColor узла Material.

Теперь, чтобы определить красный VBox и зелёный VBox, в программу после текста, напечатанного выше, необходимо добавить новый узел Vbox с полем (полями), описанными в интерфейсе (или без них – тогда значения по умолчанию) :

```
VBox { }                        #определяет красный куб

VBox {                          #определяет зелёный куб
        boxColour 0 1 0
}
```

Поля можно делать также выделенными, используя в описании интерфейса exposedField вместо field.
Этим автоматически определяется 2 события для поля – eventIn и eventOut.

Если необходимо определить прототип во внешнем файле, нужно использовать команду EXTERNPROTO. Это определение сообщает браузеру, что часть определения объекта содержится в другом файле.
Команда используется следующим образом: в главном файле используется определение объекта с помощью EXTERNPROTO, а полное PROTO содержится в другом файле.
Синтаксис EXTERNPROTO:

```
EXTERNPROTO VBox [
        field SFColor boxColour
]
"proto.wrl"
```

В proto.wrl должен содержаться заголовок VRML и определение прототипа, и больше ничего.
Если в файле содержится несколько протипов, то следует пользоваться:
`"proto.wrl#VBox"`

При использовании EXTERNPROTO нет необходимости определять значения, которые будут использоваться по умолчанию (они должны быть определены уже во внешнем файле в описании прототипа).

**Пример (файл 3-3.wrl):** определение прототипа содержится в файле proto.wrl.

```
#VRML V2.0 utf8

EXTERNPROTO VBox [
      field SFColor boxColour
]
"proto.wrl"

VBox {}           #красный куб

Transform
{
      translation 3 0 0
      children [ VBox{ boxColour 0 1 0} ]
}
```

**Описание прототипа (файл proto.wrl):**

```
#VRML V2.0 utf8

PROTO VBox [
      field SFColor boxColour 1 0 0
]
{     Shape {
            appearance Appearance {
                  material Material {
                        diffuseColor IS boxColour
                  }
            }
            geometry Box {
            }
      }
}
```

### 3.5. Связь с другими файлами при помощи узла Inline

Другой путь многократного использования VRML-кода - это использование узла **Inline**. Этот узел берёт данные из внешнего файла и вставляет их в текущий файл.

**Inline (Встраивание).**

Синтаксис узла:

```
Inline {
         exposedField       MFString  url                         []
         field              SFVec3f           bboxCenter          0 0 0
         field              SFVec3f           bboxSize            -1 -1 -1
}
```

Назначение полей:

- url – название внешнего файла.

- bboxCenter | - как в узлах Group
- bboxSize | или Anchor.

Например:

```
#если существует модель стула в файле chair.wrl, можно вставить эту модель в сцену
так:
Inline {
```

```
        url "chair.wrl"
}
#файл, используемый в Inline, должен быть VRML-файлом, написанным по всем правилам
```

**Пример (файл 3-4.wrl):** добавляются красный и зелёный куб из файла 3-3.wrl.

```
#VRML V2.0 utf8

Inline      { url "3-3.wrl" }

Transform
{
      translation 0 4 0
      children
      [
            Shape
            {
                  geometry Sphere {radius 1}
            }
      ]
}
```

### 3.6. Компрессия с помощью утилиты GZIP

Файлы с подготовленными и отлаженными программами на языке VRML можно сжимать (компрессировать) с помощью утилиты GZIP. Например:

gzip.exe f-117.wrl

Файл f-117.wrl будет сжат и переименован в f-117.wrz.

# Тема 4.

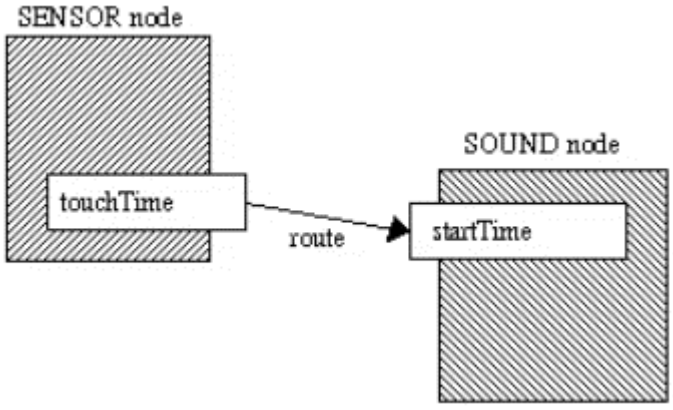
# СОЗДАНИЕ ДИНАМИЧЕСКИХ ОБЪЕКТОВ – СОБЫТИЯ, МАРШРУТЫ, СЕНСОРЫ И ИНТЕРПОЛЯТОРЫ.

**4.1. События.**

Также, как имена полей, многие узлы содержат имена событий (events).
Существует два типа событий - **eventIn** и **eventOut**.
События **eventOut** – выходящие события, которые генерируют информацию (например, изменение значения или время щелчка мыши).
События **eventIn** – входящие события, которые принимают информацию из узла снаружи и что-то делают с ними.
Каждое событие имеет тип, связанный с ним (описано выше).
Некоторые узлы содержат поля, которые являются выделенными (exposed field). Это означает, что узел имеет два события, определённых для этого поля - **set_fieldname** и **fieldname_changed.** Эти события являются событиями **eventIn** и **eventOut** для поля, которые могут быть использованы для установки значения и сообщения миру снаружи о том, когда он был изменён. Если использовать **set_fieldname** для установки значения поля, узел генерирует событие **fieldname_changed**. Для простоты использования **set_** и **_changed** части названий событий могут быть отброшены и браузер сам решит, какое событие использовать. Если поле не является выделенным, оно не может быть изменено событием, и его значение в файле – это значение, которое будет использоваться для него всё время. Для просмотра, какие поля являются выделенными, следует обратиться к описанию синтаксиса узлов.

**4.2. Маршруты.**

Для того, чтобы делать какие-то полезные действия с событиями, необходимо их связать между собой. Эта связь определяется как **ROUTE** (маршрут). Например, для связи **touchTime eventOut** с **startTime eventIn** (например, для проигрывания звука по щелчку мыши), мы направили бы событие следующим образом:




```
ROUTE SENSOR.touchTime TO SOUND.startTime
```

Эта часть кода направляет событие **touchTime** из узла TouchSensor (о нём рассказывается далее) в событие **startTime** в узле Sound. Когда TouchSensor нажат, звук начинает играть. Здесь необходимо использовать определение DEF для каждого узла, чтобы была возможность связать их (у каждого узла должно быть своё индивидуальное имя).
Сначала определяем узлы TouchSensor и Sound (с полями внутри них):

```
DEF SENSOR TouchSensor {
}
DEF SOUND Sound {
}
```

Если существует ряд объектов с одинаковыми названиями (с использованием USE), и маршруты между ними, все объекты взаимодействуют, т.е. если необходимо, чтобы только один взаимодействовал, нужно дать ему уникальное имя или использовать узел PROTO.

### 4.3. Сенсоры (датчики) - генераторы событий.

#### 4.3.1. SphereSensor (Сферический датчик).

Синтаксис узла:

```
SphereSensor {
        exposedField    SFBool          autoOffset              TRUE
        exposedField    SFBool          enabled                 TRUE
        exposedField    SFRotation      offset                  0 1 0 0
        eventOut        SFBool          isActive
        eventOut        SFRotation      rotation_changed
        eventOut        SFVec3f         trackPoint_changed
}
```

Этот датчик позволяет вращать геометрические объекты. Датчик определяет нажатие кнопки мыши, когда указатель находится на привязанной к нему геометрии. Пользователь может перетаскивать датчик (при нажатой кнопке мыши передвигать указатель), тем самым вращая объекты.
Датчик можно прикрепить к геометрическим объектам, используя их в одной группе.

Назначение полей:

- autoOffset – если значение этого поля = TRUE, то вращение объектов будет сохраняться между нажатиями, если = FALSE, то при каждом последующем нажатии объекты будут восстанавливаться в первоначальное состояние.
- enabled - датчик включен (TRUE), или датчик выключен (FALSE).
- offset - при autoOffset=TRUE значение этого поля после каждого освобождения кнопки мыши устанавливается в текущее положение объекта, при autoOffset=FALSE при каждом нажатии объект вращение объекта будет устанавливаться по значению этого поля.
- isActive – при нажатии на привязанные к датчику объекты =TRUE, иначе = FALSE.
- rotation_changed – значение поворота датчика.
- trackPoint_changed – координаты точки нажатия.

**Пример (файл 4-1.wrl):** создаются 3 геометрических объекта, 2 из которых можно вращать во всех плоскостях.

```
#VRML V2.0 utf8

Group{
      children [
            DEF SENSOR SphereSensor {
                  autoOffset TRUE
            }
            DEF TRANS Transform {
                  children [
                        Shape {
                              appearance Appearance {
                                    material Material {
                                          diffuseColor 1 1 1
                                    }
                              }
                              geometry Sphere {
                                    radius 1.5
                              }
                        }
                        Shape {
```

```
                              appearance  Appearance {
                                    material Material {
                                          diffuseColor 1 0 0
                                    }
                              }
                              geometry Cylinder {
                                    height 6
                                    radius 0.2
                              }
                        }
                  ]
            }
      ]
}

Transform {
      translation 10 0 0
      children [
            Shape {
                  appearance Appearance {
                        material Material {
                              diffuseColor 0 1 0
                        }
                  }
                  geometry Box {
                        size 4 4 4
                  }
            }
      ]
}

ROUTE SENSOR.rotation_changed TO TRANS.rotation
```

### 4.3.2. CylinderSensor (Цилиндрический датчик).

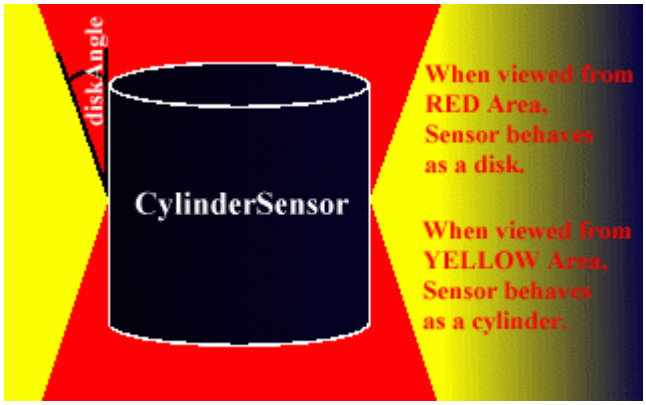
Синтаксис узла:

```
CylinderSensor {
        exposedField    SFBool          autoOffset              TRUE
        exposedField    SFFloat         diskAngle               0.262
        exposedField    SFBool          enabled                 TRUE
        exposedField    SFFloat         maxAngle                -1
        exposedField    SFFloat         minAngle                0
        exposedField    SFFloat         offset                  0
        eventOut        SFBool          isActive
        eventOut        SFRotation      rotation_changed
        eventOut        SFVec3f         trackPoint_changed
}
```

В отличии от SphereSensor, который позволяет вращать объекты вокруг любой оси, этот датчик позволяет вращать объекты только вокруг оси Y.

Назначение полей у CylinderSensor такое же, как и у SphereSensor, только появляются еще поля:

- diskAngle – когда мы смотрим на объект из красной зоны, мы вращаем датчик как диск, при нажатии передвигая указатель по кругу. Если мы смотрим из желтой зоны, то мы вращаем датчик, передвигая указатель влево или вправо.




- maxAngle | - эти поля используются, чтобы ограничить вращение цилиндра
- minAngle | (здесь указываются углы в радианах). По умолчанию эти значения такие, | что можно вращать на любой угол.

**Пример (файл 4-2.wrl):** создаются 3 геометрических объекта, 2 из которых можно вращать вокруг оси Y.

```
#VRML V2.0 utf8

Group{
	children [
		DEF SENSOR CylinderSensor {
			autoOffset TRUE
		}
		DEF TRANS Transform {
			children [
				Shape {
					appearance Appearance {
						material Material {
							diffuseColor 1 1 1
						}
					}
					geometry Cylinder {
						radius 1.5
					}
				}
				Transform
				{
				  rotation 0 0 1 1.57
				  children
				  [
				    Shape {
				      appearance  Appearance {
						material Material {
							diffuseColor 1 0 0
						}
					}
					geometry Cylinder {
						height 6
						radius 0.2
					}
				    }
				  ]
				}
			]
		}
	]
}

Transform {
	translation 10 0 0
	children [
```

```
            Shape {
                  appearance Appearance {
                        material Material {
                              diffuseColor 0 1 0
                        }
                  }
                  geometry Box {
                        size 4 4 4
                  }
            }
      ]
}

ROUTE SENSOR.rotation_changed TO TRANS.rotation
```

### 4.3.3. PlaneSensor (Плоскостной датчик) - то же, что и CylinderSensor, но в плоскости.

Синтаксис узла:

```
PlaneSensor {
        exposedField    SFBool          autoOffset              TRUE
        exposedField    SFBool          enabled                 TRUE
        exposedField    SFVec2f         maxPosition             -1 -1
        exposedField    SFVec2f         minPosition             0 0
        exposedField    SFVec3f         offset                  0 0 0
        eventOut        SFBool          isActive
        eventOut        SFVec3f         trackPoint_changed
        eventOut        SFVec3f         translation_changed
}
```

Объекты, к которым привязан этот датчик, можно двигать в плоскости, меняя локальные X и Y координаты. Локальная Z-координата считается равной 0.

Назначение полей такое же, как у SphereSensor, только появляются новые поля:

- maxPosition | - эти поля определяют диапазон движения датчика по X и Y
- minPosition | (при значениях по умолчанию датчик можно перемещать на любые
  | расстояния).

- translation_changed – новая позиция датчика после перемещения.

**Пример (файл 4-3.wrl):** красный куб можно перемещать в пределах белой поверхности.

```
#VRML V2.0 utf8

Transform {
      children [
            DEF SENSOR PlaneSensor {
                  maxPosition 2.5 2.5
                  minPosition -2.5 -2.5
            }
            DEF TRANS Transform {
                  children [
                        Shape {
                              appearance Appearance {
                                    material Material {
                                          diffuseColor 1 0 0
                                    }
                              }
                              geometry Box {
                                    size 0.5 0.5 0.5
                              }
                        }
                  ]
```

```
            }
      ]
}

Transform {
      rotation 0 0 1 1.57
      children [
            Shape {
                  appearance Appearance {
                        material Material {
                        }
                  }
                  geometry Box {
                        size 5 5 0.1
                  }
            }
      ]
}

ROUTE SENSOR.translation_changed TO TRANS.translation
```

### 4.3.4. TimeSensor (Временной датчик) – таймер.

Синтаксис узла:

```
TimeSensor {
        exposedField    SFTime          cycleInterval           1
        exposedField    SFBool          enabled                 TRUE
        exposedField    SFBool          loop                    FALSE
        exposedField    SFTime          startTime               0
        exposedField    SFTime          stopTime                0
        eventOut        SFTime          cycleTime
        eventOut        SFFloat         fraction_changed
        eventOut        SFBool          isActive
        eventOut        SFTime          time
}
```

К этому узлу не привязывается никакой геометрии.

Назначение полей:

- cycleInterval – интервал цикла (секунды).
- enabled – таймер включен (TRUE), или таймер выключен (FALSE).
- loop – при значении TRUE события будут генерироваться после каждого интервала цикла, иначе они будут генерироваться только после первого интервала цикла.

- startTime | - эти значения определяют, когда таймер включается или выключается, если
- stopTime | stopTime<=startTime, то таймер будет работать все время.

- cycleTime – это событие генерируется каждый раз, когда таймер достигает интервала цикла, и имеет значение текущего времени.
- fraction_changed – это поле генерирует непрерывный поток сигналов, необходимый для интерполяторов. Событие генерируется постоянно, и так быстро, как только может. Значение этого события интерполируется во время интервала цикла от 0 до 1 (при достижении таймером значения интервала цикла, значение fraction_changed = 1).
- isActive – генерируется, когда таймер начинает работать или останавливается (TRUE – таймер запущен, FALSE-прекратил работу).
- time – генерируется так же, как и fraction_changed, только имеет другой тип.

Пример работы узла TimeSensor будет продемонстрирован в примере по узлу ColorInterpolator.

### 4.3.5. TouchSensor (Датчик прикосновения).

Синтаксис узла:

```
TouchSensor {
        exposedField    SFBool          enabled                 TRUE
        eventOut        SFVec3f         hitNormal_changed
        eventOut        SFVec3f         hitPoint_changed
        eventOut        SFVec2f         hitTexCoord_changed
        eventOut        SFBool          isActive
        eventOut        SFBool          isOver
        eventOut        SFTime          touchTime
}
```

Датчик прикосновения можно привязать к геометрическим объектам, описав их в одной группе. Например, чтобы привязать к сфере датчик прикосновения, можно использовать следующий код:

```
Group {
        children [
                DEF TOUCH TouchSensor {}
                Shape {
                        appearance Appearance {
                                material Material { }
                        }
                        geometry Sphere { }
                }
        ]
}
```

Назначение полей:

- enabled - датчик включен (TRUE), или датчик выключен (FALSE).
- hitNormal_changed – посылает при нажатии нормаль поверхности в точке, над которой находится указатель.
- hitPoint_changed – координаты точки, над которой находится указатель.
- hitTexCoord_changed – координаты текстуры в точке, над которой указатель.
- isActive – при нажатии на объекты это событие = TRUE.
- isOver – это событие = TRUE, когда указатель находится над геометрическими объектами, к которым прикреплен датчик.
- touchTime – время нажатия на объекты, к которым прикреплен датчик (генерируется при отпускании кнопки мыши).

**Пример (файл 4-4.wrl):** за указателем, перемещающимся по зеленой сфере большого диаметра, закреплена красная сфера маленького диаметра.

```
#VRML V2.0 utf8

Group {
      children [
            DEF TOUCH TouchSensor {}
            Shape {
                  appearance Appearance {
                        material Material {
                              diffuseColor 0 1 0
                        }
                  }
                  geometry Sphere {
                        radius 3.0
                  }
            }
      ]
}

DEF L_SPHERE Transform {
```

```
      children [
            Shape {
                  appearance Appearance {
                        material Material {
                              diffuseColor 1 0 0
                        }
                  }
                  geometry Sphere {
                        radius 0.1
                  }
            }
      ]
}

ROUTE TOUCH.hitPoint_changed TO L_SPHERE.translation
```

### 4.3.6. VisibilitySensor (Датчик видимости).

Синтаксис узла:

```
VisibilitySensor {
        exposedField    SFVec3f         center                  0 0 0
        exposedField    SFBool          enabled                 TRUE
        exposedField    SFVec3f         size                    0 0 0
        eventOut        SFTime          enterTime
        eventOut        SFTime          exitTime
        eventOut        SFBool          isActive
}
```

Этот узел определяет невидимый параллелепипед в области сцены, и при нахождении этого параллелепипеда в поле зрения генерируются события.

Назначение полей:

- center – центр параллелепипеда.
- enabled - датчик включен (TRUE), или датчик выключен (FALSE).
- size – размеры параллелепипеда.
- enterTime – событие генерируется при нахождении параллелепипеда в поле зрения, и определяет время входа.
- exitTime – время выхода параллелепипеда из поля зрения (генерируется при выходе).
- isActive – при попадании параллелепипеда в поле зрения значение этого события = TRUE, иначе его значение = FALSE.

Пример работы узла VisibilitySensor будет продемонстрирован в примере работы узла ColorInterpolator.

### 4.3.7. ProximitySensor (Датчик приближения) - регион в пространстве чувствительный к проникновению в него.

Синтаксис узла:

```
ProximitySensor {
        exposedField    SFVec3f         center                  0 0 0
        exposedField    SFVec3f         size                    0 0 0
        exposedField    SFBool          enabled                 TRUE
        eventOut        SFBool          isActive
        eventOut        SFVec3f         position_changed
        eventOut        SFRotation      orientation_changed
        eventOut        SFTime          enterTime
        eventOut        SFTime          exitTime
}
```

Этот узел работает так же, как VisibilitySensor, за исключением того, что события генерируются при каждом входе или выходе из определенного параллелепипеда.

Здесь еще существует два дополнительных события, которые генерируются только в том случае, если пользователь находится внутри параллелепипеда:

- position_changed – при смене позиции пользователя здесь содержится новое значение позиции.
- orientation_changed – при смене ориентации здесь новое значение ориентации.

**Пример (файл 4-5.wrl):** если подойти к синему цилиндру и поворачиваться в различные стороны, цилиндр тоже будет поворачиваться на такой же угол.

```
#VRML V2.0 utf8

Group {
      children [
            DEF PROXIMITY ProximitySensor {size 10 10 10}
            DEF CYLINDER Transform
            {
              children [ Transform {
                translation 0 -0.5 -3
                rotation 1 0 0 1.57
                children
                [
                  Shape {
                        appearance Appearance {
                              material Material {
                                    diffuseColor 0 0 1
                              }
                        }
                        geometry Cylinder {
                              radius 0.1
                              height 3
                        }
                  }
                ]
              }
              ]
            }
      ]
}

Transform {
translation 0 -0.7 -2.5
      children [
            Shape {
                  appearance Appearance {
                        material Material {
                              diffuseColor 1 1 1
                        }
                  }
                  geometry Box {
                        size 1 0.3 0.3
                  }
            }
      ]
}

ROUTE PROXIMITY.orientation_changed TO CYLINDER.rotation
```

### 4.4. Интерполяторы по таблице значений.

#### 4.4.1. ColorInterpolator (Интерполятор цвета).

Синтаксис узла:

```
ColorInterpolator {
        eventIn         SFFloat         set_fraction
        exposedField    MFFloat         key                     []
        exposedField    MFColor         keyValue                []
        eventOut        SFColor         value_changed
}
```

Назначение полей:

- keyValue – набор значений цветов. Линейная интерполяция происходит между этими значениями.
- key – для каждого значения из keyValue здесь записывается значение от 0 до 1.
- set_fraction - входное событие, значение которого сравнивается со значениями из key, и при совпадении осуществляется интерполяция между следующими двумя значениями из keyValue.
- value_changed – выходное событие - текущее значение цвета, которое интерполируется между значениями цветов из keyValue.

**Пример (файл 4-6.wrl):** интерполируется цвет цилиндра (полный цикл интерполяции – 5 секунд – используется узел TimeSensor), если белый куб находится в поле видимости (используется узел VisibilitySensor).

```
#VRML V2.0 utf8
Transform
{
    translation 5 0 0
    children [
      DEF VIS VisibilitySensor
      {
            size 0.5 0.5 0.5
      }
      Shape
      {
            geometry Box { size 0.5 0.5 0.5 }
      }
    ]
}
DEF TIME TimeSensor
{
      enabled FALSE
      cycleInterval 5
      loop TRUE
}
DEF COL_INT ColorInterpolator
{
      key [0, 0.33, 0.66, 1]
      keyValue[1 0 0, 0 1 0, 0 0 1, 1 0 0]
}
Shape {
      appearance Appearance {
            material DEF MATERIAL Material {
            }
      }
      geometry Cylinder {
      }
}
ROUTE VIS.isActive TO TIME.enabled
ROUTE TIME.fraction_changed TO COL_INT.set_fraction
ROUTE COL_INT.value_changed TO MATERIAL.diffuseColor
```

### 4.4.2. CoordinateInterpolator (Интерполятор координат).

Синтаксис узла:

```
CoordinateInterpolator {
        eventIn         SFFloat         set_fraction
        exposedField    MFFloat         key                     []
        exposedField    MFVec3f         keyValue                []
        eventOut        MFVec3f         value_changed
}
```

Принцип, как и в ColorInterpolator, только здесь происходит интерполяция между значениями координат.

**Пример (файл 4-7.wrl):** интерполируются координаты пирамиды.

```
#VRML V2.0 utf8

DEF TIME TimeSensor
{
      cycleInterval 5
      loop TRUE
}


DEF COORD_INT CoordinateInterpolator {
      key [0, 0.2, 0.4, 0.6, 0.8, 1]
      keyValue [   0 1 1, 0 1 -1, -1 -1 0, 1 -1 0,
                   0 2 2, 0 1 -1, -1 -1 0, 1 -1 0,
                   0 1 1, 0 2 -2, -1 -1 0, 1 -1 0,
                   0 1 1, 0 1 -1, -2 -2 0, 1 -1 0,
                   0 1 1, 0 1 -1, -1 -1 0, 2 -2 0,
                   0 1 1, 0 1 -1, -1 -1 0, 1 -1 0 ]
}


Transform {
      translation 0 3 -10
      children [
            Shape {
                  appearance Appearance {
                        material Material {
                              diffuseColor 0 0 1
                        }
                  }
                  geometry IndexedFaceSet {
                        coord DEF COORD Coordinate {
                              point [ 0 1 1, 0 1 -1, -1 -1 0, 1 -1 0]
                        }
                        coordIndex [      2 3 0 -1,
                                          2 0 1 -1,
                                          3 2 1 -1,
                                          3 1 0 -1 ]
                  }
            }
      ]
}

ROUTE TIME.fraction_changed TO COORD_INT.set_fraction
ROUTE COORD_INT.value_changed TO COORD.point
```

### 4.4.3. NormalInterpolator (Интерполятор векторов нормалей).

Синтаксис узла:

```
NormalInterpolator {
        eventIn         SFFloat         set_fraction
        exposedField    MFFloat         key                     []
        exposedField    MFVec3f         keyValue                []
        eventOut        MFVec3f         value_changed
}
```

Этот узел полностью совпадает с CoordinateInterpolator.

**Пример (файл 4-8.wrl):** меняется освещение пирамиды.

```
#VRML V2.0 utf8

DEF TIME TimeSensor
{
      cycleInterval 5
      loop TRUE
}

DEF NORM_INT NormalInterpolator {
      key [0, 0.25, 0.5, 0.75, 1]
      keyValue [ 0 0 1, 1 0 0, 0 0 -1, -1 0 0,
               1 0 0, 0 0 -1, -1 0 0, 0 0 1,
               0 0 -1, -1 0 0, 0 0 1, 1 0 0,
               -1 0 0, 0 0 1, 1 0 0, 0 0 -1,
               0 0 1, 1 0 0, 0 0 -1, -1 0 0 ]
}


Transform {
      translation 0 3 -10
      children [
            Shape {
                  appearance Appearance {
                        material Material {
                              diffuseColor 1 0 0
                        }
                  }
                  geometry IndexedFaceSet {
                        coord Coordinate {
                              point [ 0 1 1, 0 1 -1, -1 -1 0, 1 -1 0]
                        }
                        coordIndex [      2 3 0 -1,
                                    2 0 1 -1,
                                    3 2 1 -1,
                                    3 1 0 -1 ]
                        normal DEF NORMAL Normal {
                              vector [ 0 0 1,
                                     1 0 0,
                                     0 0 -1,
                                     -1 0 0]
                        }
                        normalPerVertex FALSE
                        normalIndex [ 0 1 2 3]
                  }
            }
      ]
}

ROUTE TIME.fraction_changed TO NORM_INT.set_fraction
ROUTE NORM_INT.value_changed TO NORMAL.vector
```

### 4.4.4. OrientationInterpolator (Интерполятор ориентации).

Синтаксис узла:

```
OrientationInterpolator {
        eventIn         SFFloat         set_fraction
        exposedField    MFFloat         key                     []
        exposedField    MFRotation      keyValue                []
        eventOut        SFRotation      value_changed
}
```

**Пример (файл 4-9.wrl):** вокруг сферы вращается конус.

```
#VRML V2.0 utf8

DEF TIME TimeSensor
{
      cycleInterval 5
      loop TRUE
}

DEF O_INT OrientationInterpolator
{
      key [0, 0.5, 1]
      keyValue[0 0 1 0, 0 0 1 -3.14, 0 0 1 -6.28]
}

Shape {
      appearance Appearance {
            material Material {
                  diffuseColor 1 1 1
            }
      }
      geometry Box {
      }
}

DEF TRANS Transform
{
children
[
Transform
{
      translation 0 -3 0
      children
      [
            Shape {
                  appearance Appearance {
                        material Material {
                              diffuseColor 1 0 0
                        }
                  }
                  geometry Cone {
                  }
            }
      ]
}
]
}

ROUTE TIME.fraction_changed TO O_INT.set_fraction
ROUTE O_INT.value_changed TO TRANS.rotation
```

### 4.4.5. PositionInterpolator (Интерполятор позиции).

Синтаксис узла:

```
PositionInterpolator {
        eventIn         SFFloat         set_fraction
        exposedField    MFFloat         key                     []
        exposedField    MFVec3f         keyValue                []
        eventOut        SFVec3f         value_changed
}
```

**Пример (файл 4-10.wrl):** сфера двигается вокруг куба.

```
#VRML V2.0 utf8

DEF TIME TimeSensor
{
      cycleInterval 5
      loop TRUE
}

DEF POS_INT PositionInterpolator
{
      key [0, 0.2, 0.4, 0.6, 0.8, 1]
      keyValue[-3 -3 0, -3 3 0, 3 3 0, 3 -3 0, -3 -3 0]
}

Shape {
      appearance Appearance {
            material Material {
                  diffuseColor 1 1 1
            }
      }
      geometry Box {
            size 1 1 1
      }
}

DEF TRANS Transform
{
      translation -3 -3 0
      children
      [
            Shape {
                  appearance Appearance {
                        material Material {
                              diffuseColor 1 0 0
                        }
                  }
                  geometry Sphere {
                        radius 0.25
                  }
            }
      ]
}

ROUTE TIME.fraction_changed TO POS_INT.set_fraction
ROUTE POS_INT.value_changed TO TRANS.translation
```

### 4.4.6. ScalarInterpolator (Интерполятор любого значения с плавающей точкой).

Синтаксис узла:

```
ScalarInterpolator {
        eventIn         SFFloat         set_fraction
        exposedField    MFFloat         key                     []
        exposedField    MFFloat         keyValue                []
        eventOut        SFFloat         value_changed
}
```

**Пример (файл 4-11.wrl):** интерполируется прозрачность куба.

```
#VRML V2.0 utf8

DEF TIME TimeSensor
{
        cycleInterval 5
        loop TRUE
}

DEF SC_INT ScalarInterpolator
{
        key [0, 0.5, 1]
        keyValue[0.1, 0.9, 0.1]
}

Shape {
        appearance Appearance {
                material DEF MATERIAL Material {
                        diffuseColor 0 1 0
                        transparency 0.1
                }
        }
        geometry Box {
                size 1 1 1
        }
}

Transform
{
        translation 0 0 -5
        children
        [
                Shape {
                        appearance Appearance {
                                material Material {
                                        diffuseColor 1 1 1
                                }
                        }
                        geometry Sphere {
                                radius 1
                        }
                }
        ]
}

ROUTE TIME.fraction_changed TO SC_INT.set_fraction
ROUTE SC_INT.value_changed TO MATERIAL.transparency
```

# Тема 5.

## Навигация и освещение.

### 5.1. Связываемые узлы.

Некоторые узлы, которые воздействуют на общее представление и режим навигации мира, являются связываемыми. Когда узел связан, установки внутри него воздействуют на состояние мира. Когда узел несвязан, другой узел того же самого типа должен быть связан, так установки в этом другом узле воздействуют на мир. Таким образом, можно переключаться между различными установками для мира, выбирая, какой из узлов будет связан, и, соответственно, будет воздействовать на мир.

Связываемыми узлами являются следующие:

- **Background**
- **Fog**
- **NavigationInfo**
- **Viewpoint**

### 5.1.1.Связывающийся стек.

Браузер поддерживает 4 структуры для внутренних работ, каждая для своего типа связываемых узлов. Они называются связывающими стеками. Когда браузером анализируется VRML файл, первый узел соответствующего типа помещен в верхнюю часть стека, который соответствует его типу. Последующие узлы добавляются в нижнюю часть стека. Текущий связанный узел (то есть тот, который является активным) – это тот, который находится в верхней части стека (остальные узлы не активны). Как только мир загружен, мы можем управлять стеком при помощи некоторого набора событий.

### 5.1.2. Связывание узлов.

Все связываемые узлы имеют 2 события - set_bind eventIn и isBound eventOut. У обоих значения типа SFBOOL. Когда событие set_bind со значением TRUE посылается узлу, этот узел перемещается в верхнюю часть стека, и становится связанным узлом. Затем этот узел пошлет событие isBound со значением TRUE. Узел, который был в верхней части стека, посылает isBound=FALSE в этот же момент. Здесь есть одно исключение - если узел уже в верхней части стека, ничего не случается, и никакие события не посылаются.

Если Вы посылаете set_bind=FALSE к узлу, это удаляет его из стека навсегда. Если это связанный узел, то посылается isBound=FALSE, и следующий узел в стеке делается связанным узлом. Если нет, узел просто удаляется, и никаких событий не посылается. Если послать событие set_bind узлу, который был удален из стека, то ничего не произойдет.

Событие bindTime – здесь посылается время, когда узел становится связанным, или перестает быть связанным.

### 5.1.3. Использование связывания.

Теперь, когда известно, как использовать связывание, что можно сделать полезного? Можно менять фон и стили тумана. Также, используя различные узлы NavigationInfo, можно изменять размеры аватара пользователя, менять тип навигации, и т.д.

Однако, наиболее полезный узел - Viewpoint. Если связать пользователя с точкой наблюдения, он будет перемещаться туда немедленно в стиле, определенном в узле (поле jump). Таким образом, можно получить сценарии перемещения пользователя во всем мире, связывая пользователя с различными точками наблюдения.

## 5.2. Навигация.

### 5.2.1. Viewpoint (Точка наблюдения) .

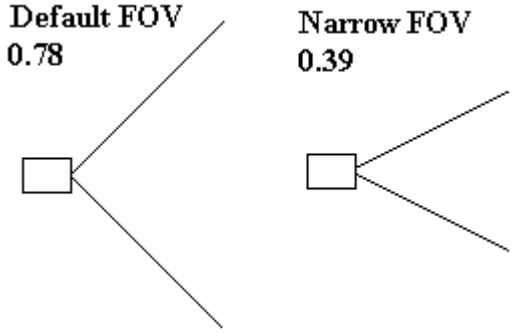
Синтаксис узла:

```
Viewpoint {
        eventIn         SFBool          set_bind
        exposedField    SFFloat         fieldOfView             0.785398
        exposedField    SFBool          jump                    TRUE
        exposedField    SFRotation      orientation             0 0 1 0
        exposedField    SFVec3f         position                0 0 10
        field           SFString        description             ""
        eventOut        SFTime          bindTime
        eventOut        SFTime          isBound
}
```

С помощью этого узла можно расставить точки наблюдения в сцене, а также установить параметры наблюдения.

Назначение полей:

- set_bind – см. раздел 5.1.
- fieldOfView – угол обзора камеры в радианах (его также можно поменять, если использовать узел Viewpoint внутри узла Transform).




- jump – при использовании события set_bind определяет способ перехода.
- orientation – начальная ориентация камеры.
- position – начальная позиция камеры.
- description – описание камеры (выводится в браузере).
- bindTime – посылает время, когда узел становится связанным, или перестает быть связанным.
- isBound - см. раздел 5.1.

**Пример (файл 5-1.wrl):** иллюстрирует действие входящего события set_bind для “Камера 2”. При нажатии на TouchSensor, прикрепленный к красной сфере происходит переключение на “Камера 2”, при отпускании TouchSensor становится активной “Камера 1”.

```
#VRML V2.0 utf8
# зеленая сфера
Transform
{
  translation 0 0 10
  children [
        Shape {
                appearance Appearance {
                        material Material {
                                diffuseColor 0 1 0
                        }
                }
                geometry Sphere {
                }
        }
```

```
  ]
}
# маленькая красная сфера, к которой прикреплен TouchSensor
Transform
{
  translation 0 3 5
  children [
        DEF TS TouchSensor
        {
        }
        Shape {
                appearance Appearance {
                        material Material {
                                diffuseColor 1 0 0
                        }
                }
                geometry Sphere {
                        radius 0.2
                }
        }
  ]
}
# синий куб
Shape {
        appearance Appearance {
                material Material {
                        diffuseColor 0 0 1
                }
        }
        geometry Box {
        }
}
# Камера 1 (активная по умолчанию)
Viewpoint
{
        position 0 2.5 -3
        orientation 0 1 0 3.14
        description "Камера 1"
}
# Камера 2
DEF VP Viewpoint
{
        position  0 2.5 19
        description "Камера 2"
        fieldOfView 0.39
}
# по нажатию на TouchSensor камера №2 перемещается в вершину
# связывающего стека
ROUTE TS.isActive TO VP.set_bind
```

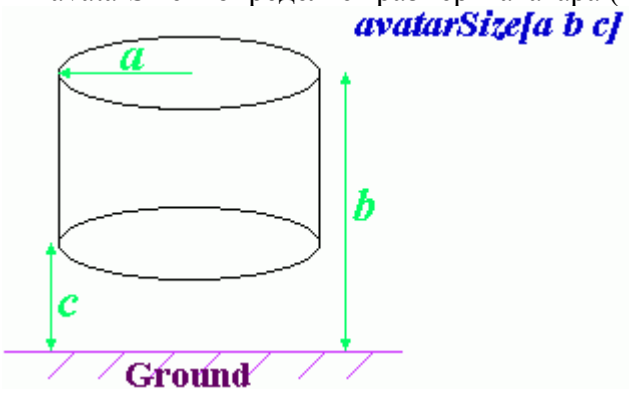
### 5.2.2. NavigationInfo (Информация о навигации).

Синтаксис узла:

```
NavigationInfo {
        eventIn         SFBool          set_bind
        exposedField    MFFloat         avatarSize              [0.25, 1.6, 0.75]
        exposedField    SFBool          headlight               TRUE
        exposedField    SFFloat         speed                   1.0
        exposedField    MFString        type                    "WALK"
        exposedField    SFFloat         visibilityLimit         0.0
        eventOut        SFBool          isBound
}
```

Назначение полей:

- set_bind - см. раздел 5.1.
- avatarSize – определяет размеры аватара (представления пользователя) пользователя.




a – размер по горизонтали - влияет на столкновение с другими объектами.
b – размер по вертикали – определяет, насколько высоко над объектами находится позиция наблюдения.
c – определяет, насколько высокие объекты можно “перешагнуть”.
MS VRML Viewer игнорирует эти параметры, а также speed, visibilityLimit.

- headlight – подсветка для сцены (при значении FALSE освещение по умолчанию выключено, необходимо использовать свои источники освещения).
- speed – скорость перемещения по миру (м/с). При значении этого параметра =0 получится только крутится на месте.
- type – тип навигации, который будет установлен в браузере (какими кнопками можно будет пользоваться при просмотре). Можно указывать несколько типов навигации.
  Возможные значения параметра:
  "ANY" – панель управления браузера включена целиком.
  "WALK" – только ходьба.
  "EXAMINE" – только изучение (движением мыши вращается вся сцена).
  "FLY" – только полет (отличается от ходьбы отсутствием гравитации).
  "NONE" – целиком выключается панель управления браузера.
- visibilityLimit – определяет, как далеко аватар может видеть. Рендеринг за пределами этого значения браузер не проводит. Значение по умолчанию соответствует бесконечному пределу.
- isBound - см. раздел 5.1.

**Пример (файл 5-2.wrl):** можно использовать только режимы навигации “EXAMINE” и “WALK”. Выключена подсветка по умолчанию для мира.

```
#VRML V2.0 utf8
NavigationInfo
{
      type ["EXAMINE","WALK"]
      # ^^^ определяем возможные типы навигации
}
# зеленая сфера
Transform
{
  translation 0 0 10
  children [
      Shape {
            appearance Appearance {
                  material Material {
                        diffuseColor 0 1 0
                  }
            }
            geometry Sphere {
```

```
            }
      }
  ]
}

# синий куб
Shape {
      appearance Appearance {
            material Material {
                  diffuseColor 0 0 1
            }
      }
      geometry Box {
      }
}

# Камера
Viewpoint
{
      position 11 1.5 5
      orientation 0 1 0 1.57
      description "Камера"
}
```

### 5.2.3. WorldInfo (Информация о мире).

Синтаксис узла:

```
WorldInfo {
        field           MFString        info                    []
        field           SFString        title                   ""
}
```

Назначение полей:

- info – строки с информацией.
- title – название мира.

**Пример:** в примере по узлу Collision.

### 5.2.4. Collision (Столкновение).

Синтаксис узла:

```
Collision {
        eventIn         MFNode          addChildren
        eventIn         MFNode          removeChildren
        exposedField    MFNode          children                []
        exposedField    SFBool          collide                 TRUE
        field           SFVec3f         bboxCenter              0 0 0
        field           SFVec3f         bboxSize                -1 -1 -1
        field           SFNode          proxy                   NULL
        eventOut        SFTime          collideTime
}
```

Этот узел в качестве выходного события возвращает время столкновения аватара с объектами, а также регулирует, пройдет аватар сквозь объекты или нет.

Назначение полей:

- addChildren | - это входные события соответственно для добавления/удаления
- removeChildren | потомков.

- children – потомки (объекты).

- collide – при значении TRUE нельзя проходить сквозь объекты, при значении FALSE можно.
- bboxCenter | - эти 2 поля определяют центр и размеры параллелепипеда, в пределах

- bboxSize | которого существуют дочерние узлы. Это может использоваться с целью оптимизации сцены. Результат не определен, если в размеры параллелепипеда не влезает хотя бы один из потомков. Значение по умолчанию этих двух узлов подразумевает, что параллелепипед не существует.
- proxy – объект, указанный здесь, не отображается, но время столкновения с ним генерируется. Это может быть полезно для разгрузки браузера от вычислений (здесь можно указать объект более простой – например, если нужно зарегистрировать столкновение с объектом “звезда”, то можно указать объект “сфера”, внутри которой будет “звезда”, и регистрировать столкновение с объектом “сфера”).
- collideTime – время столкновения.

**Пример (файл 5-3.wrl):** через объекты можно проходить насквозь.

```
#VRML V2.0 utf8

WorldInfo {
        title "Пример по узлу Collision"
        info ["(C) Центр Компьютерного Интерактивного Моделирования",
                "при СПбГУАП"]
}


Collision
{
collide FALSE  # столкновений с объектами не происходит
children
[
        # зеленая сфера
        Transform
        {
          translation 0 0 10
          children [
                Shape {
                        appearance Appearance {
                                material Material {
                                        diffuseColor 0 1 0
                                }
                        }
                        geometry Sphere {
                        }
                }
          ]
        }

        # синий куб
        Shape {
                appearance Appearance {
                        material Material {
                                diffuseColor 0 0 1
                        }
                }
                geometry Box {
                }
        }
]
}
# Камера
Viewpoint
{
        position 11 0.5 5
        orientation 0 1 0 1.57
        description "Камера"
}
```

### 5.2.5. LOD (Уровень детализации).

Синтаксис узла:

```
LOD {
        exposedField    MFNode          level                   []
        field           SFVec3f         center                  0 0 0
        field           MFFloat         range                   []
}
```

При помощи этого узла можно разгрузить браузер от лишних вычислений.

Назначение полей:

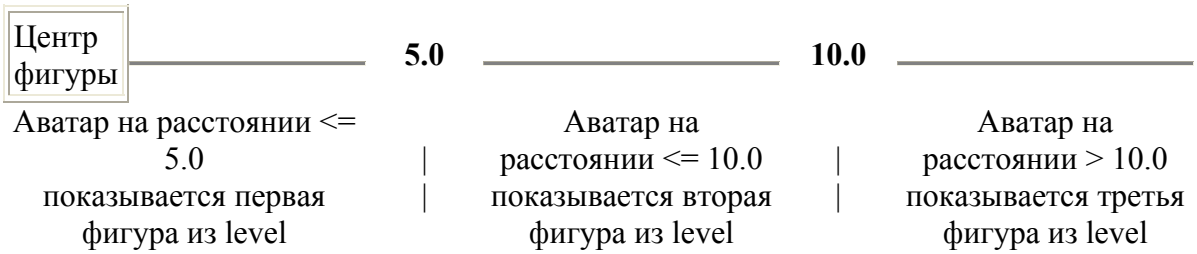
- level – здесь указываются последовательно фигуры, которые будут отображаться при отходе на определённые расстояния (указанные в поле range) от координаты, указанной в поле center.
- center – здесь указывается центр фигур.
- range – расстояния (количество расстояний = количество фигур - 1).

Проиллюстрировать это можно так:

Например,

range [ 5.0, 10.0 ]

тогда




**Пример (файл 5-4.wrl):** меняются уровни детализации объекта.

```
#VRML V2.0 utf8

LOD {
    center 0.0 0.0 0.0
    range [ 20.0, 40.0 ]
    level [
    # High-detail
        Inline { url "5-4-1.wrl" },
    # если мы находимся в пределах 20 метров от центра,
    # выводится фигура из файла 5-4-1.wrl

    # Medium-detail
        Inline { url "5-4-2.wrl" },
    # если мы отходим на расстояние более 20 метров от центра,
    # выводится фигура из файла 5-4-2.wrl

    # Low-detail
        Inline { url "5-4-3.wrl" }
    # если мы отходим на расстояние более 40 метров от центра,
    # выводится фигура из файла 5-4-3.wrl
    ]
}
```

### 5.3. Освещение.

#### 5.3.1. DirectionalLight (Направленное освещение).

Синтаксис узла:

```
DirectionalLight {
        exposedField    SFFloat         ambientIntensity        0
        exposedField    SFColor         color                   1 1 1
        exposedField    SFVec3f         direction               0 0 -1
        exposedField    SFFloat         intensity               1
        exposedField    SFBool          on                      TRUE
}
```

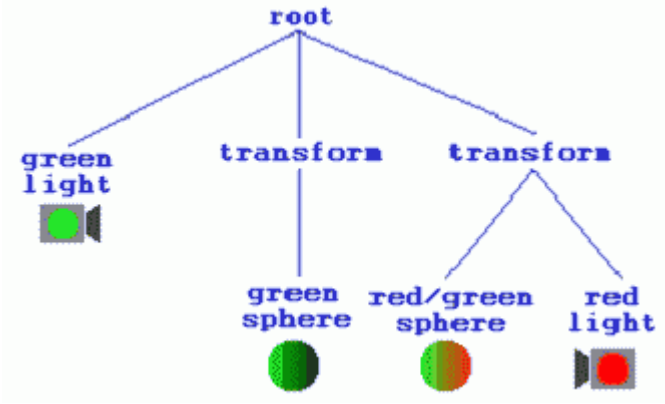
При помощи этого узла задается освещение параллельными лучами в указанном направлении. По умолчанию направление. Источник предполагается бесконечно удаленным, поэтому задается только направление освещения.

Назначение полей:

- ambientIntensity – доля источника в общем освещении сцены за счет отражения и рассеяния от объектов (от 0 до 1).
- color – цвет освещения.
- direction – вектор направления освещения.
- intensity – интенсивность освещения (от 0 до 1).
- on – источник света включен (при значении TRUE) или выключен (при значении FALSE).

По крайней мере, в MS VRML Viewer, DirectionalLight не освещает объекты, имеющие свой цвет, кроме оттенков серого.

Размещение данного узла в группирующем узле равносильно его воздействию только на объекты внутри этого группирующего узла.
По графу сцены это можно проиллюстрировать, например, так:




**Пример (файл 5-5.wrl):** освещается только сфера сверху красным светом. Подсветка для сцены отключена.

```
#VRML V2.0 utf8

# сфера
Transform
{
  translation 0 0 10
  children [
        DirectionalLight { #группируем DirectionalLight со сферой
                ambientIntensity 0.75
                intensity 1
                color 1 0 0
                direction 0 -1 0
        }
```

```
        Shape {
                appearance Appearance {
                        material Material {
                                diffuseColor 1 1 1
                        }
                }
                geometry Sphere {
                }
        }
 ]
}

# куб
Shape {
        appearance Appearance {
                material Material {
                        diffuseColor 1 1 1
                }
        }
        geometry Box {
        }
}

# Камера
Viewpoint
{
        position 11 1.5 5
        orientation 0 1 0 1.57
        description "Камера"
}
```

**5.3.2. PointLight (Точечное освещение).**

Синтаксис узла:

```
PointLight {
        exposedField    SFFloat         ambientIntensity        0
        exposedField    SFVec3f         attenuation             1 0 0
        exposedField    SFColor         color                   1 1 1
        exposedField    SFFloat         intensity               1
        exposedField    SFVec3f         location                0 0 0
        exposedField    SFBool          on                      TRUE
        exposedField    SFFloat         radius                  100
}
```

С помощью этого узла задается точечный источник света, который излучает во всех направлениях.

Назначение полей:

- ambientIntensity - доля источника в общем освещении сцены за счет отражения и рассеяния от объектов (от 0 до 1).
- attenuation – этот параметр определяет, как быстро будет падать интенсивность освещения при удалении от местоположения источника. Три числа, указываемые в качестве значения этого поля , используются вычисления интенсивности на расстоянии r от центра по формуле:

$$I_r = \frac{I_0}{\max(1, attenuation_0 + attenuation_1 r + attenuation_2 r^2)}$$

- color - цвет освещения.
- intensity - интенсивность освещения (от 0 до 1).
- location – координаты местоположения источника.
- on - источник света включен (при значении TRUE) или выключен (при значении FALSE).

- radius – радиус сферы освещения.

В MS VRML Viewer этот узел также не освещает раскрашенные объекты (кроме оттенков серого).

**Пример (файл 5-6.wrl):** при нажатии на TouchSensor, который прикреплен к зеленой сфере, включается освещение точечного источника. При отпускании TouchSensor освещение выключается.

```
#VRML V2.0 utf8
# источник, сфера и TouchSensor
Transform
{
  translation 0 3 0
  children [
        DEF PL PointLight {
                ambientIntensity 0.75
                intensity 1
                color 0 1 0
                location 0 0 0
                on FALSE
        }
        DEF TS TouchSensor
        {
        }
        Shape {
                appearance Appearance {
                        material Material {
                                diffuseColor 0 1 0
                        }
                }
                geometry Sphere {
                        radius 0.2
                }
        }
 ]
}
# сфера
Shape {
        appearance Appearance {
                material Material {
                }
        }
        geometry Sphere {
        }
}
# по нажатию на TouchSensor включается освещение
ROUTE TS.isActive TO PL.on
```

### 5.3.3. SpotLight (Прожекторное освещение).

Синтаксис узла:

```
SpotLight {
        exposedField    SFFloat         ambientIntensity        0
        exposedField    SFVec3f         attenuation             1 0 0
        exposedField    SFFloat         beamWidth               1.570796
        exposedField    SFColor         color                   1 1 1
        exposedField    SFFloat         cutOffAngle             0.785398
        exposedField    SFVec3f         direction               0 0 -1
        exposedField    SFFloat         intensity               1
        exposedField    SFVec3f         location                0 0 0
        exposedField    SFBool          on                      TRUE
        exposedField    SFFloat         radius                  100
}
```

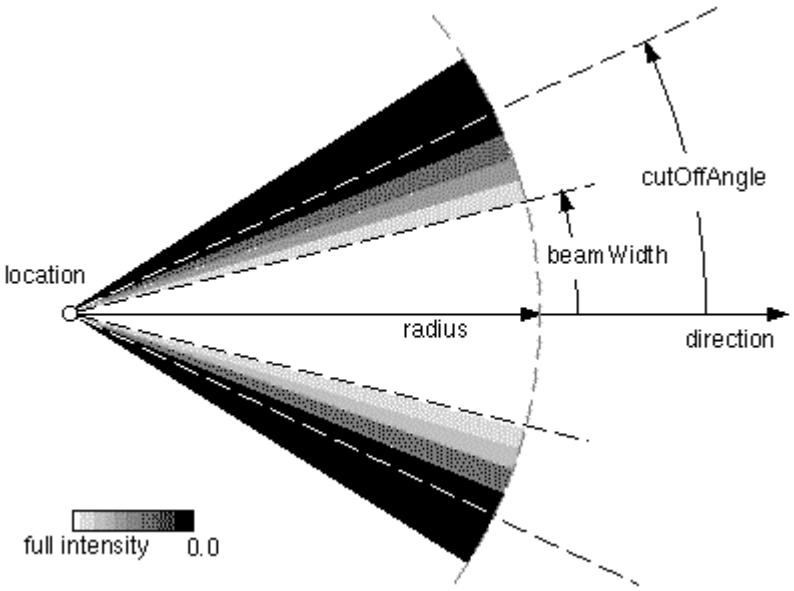
При помощи этого узла можно определить источник освещения, который имеет свое местоположение и светит в определенном направлении.

Назначение полей:

- ambientIntensity - доля источника в общем освещении сцены за счет отражения и рассеяния от объектов (от 0 до 1).
- attenuation – затухание света при удалении от источника (см. узел PointLight).
- color - цвет освещения.
- direction - вектор направления освещения.
- intensity - интенсивность освещения (от 0 до 1).
- location - координаты местоположения источника.
- on - источник света включен (при значении TRUE) или выключен (при значении FALSE).
- radius – радиус сферы освещения.
- beamWidth |– при помощи этих двух параметров можно задать размывание по краям
- cutOffAngle | светового пятна. Для этого придуманы два конуса с углами раствора beamWidth и cutoffAngle. Во внутреннем конусе (с углом beamWidth) интенсивность в направлении перпендикулярно лучу постоянна, и равна параметру intensity. Снаружи внешнего конуса (с углом cutoffAngle) интенсивность равна 0, а в зазоре между ними интенсивность линейно интерполируется. По умолчанию выставлено beamWidth > cutoffAngle, что дает пятно с неразмытыми краями.

При размещении SpotLight внутри Transform, параметры scale и translation узла Transform повлияют на все параметры самого SpotLight.

В MS VRML Viewer этот узел также не освещает раскрашенные объекты (кроме оттенков серого).




**Пример (файл 5-7.wrl):** освещаем ElevationGrid.

```
#VRML V2.0 utf8

# источник, сфера и TouchSensor
Transform
{
  translation 17.5 20 15
```

```
  children [
        DEF PL SpotLight {
                ambientIntensity 0.75
                intensity 1
                color 1 0 0
                location 0 0 0
                direction 0 -1 0
                on FALSE
                beamWidth 0.157
                cutOffAngle 0.471
        }

        DEF TS TouchSensor
        {
        }

        Shape {
                appearance Appearance {
                        material Material {
                                diffuseColor 1 0 0
                        }
                }
                geometry Sphere {
                        radius 1
                }
        }
 ]
}

# поверхность
Shape{
                appearance Appearance {
                        material Material {
                                diffuseColor 1 1 1
                        }
                }
geometry ElevationGrid {
        xDimension 7
        zDimension 6
        height [1.5, 1, 0.5, 0.5, 1, 1.5,0,
                1, 0.5, 0.25, 0.25, 0.5, 1,0,
                0.5, 0.25, 0, 0, 0.25, 0.5,0,
                0.5, 0.25, 0, 0, 0.25, 0.5,0,
                1, 0.5, 0.25, 0.25, 0.5, 1,0,
                1.5, 1, 0.5, 0.5, 1, 1.5,0]
        xSpacing 5.0
        zSpacing 5.0
        solid FALSE
        }
}

# Камера
Viewpoint
{
        position 60 10 15
        orientation 0 1 0 1.57
        description "Камера"
}

# по нажатию на TouchSensor включается освещение
ROUTE TS.isActive TO PL.on
```

# Тема 6.

## Спецэффекты и звук

### 6.1. Спецэффекты.

#### 6.1.1. Background (Фон).

Полный синтаксис узла:

```
Background {
        eventIn         SFBool          set_bind
        exposedField    MFFloat         groundAngle             []
        exposedField    MFColor         groundColor             []
        exposedField    MFString        backUrl                 []
        exposedField    MFString        bottomUrl               []
        exposedField    MFString        frontUrl                []
        exposedField    MFString        leftUrl                 []
        exposedField    MFString        rightUrl                []
        exposedField    MFString        topUrl                  []
        exposedField    MFFloat         skyAngle                []
        exposedField    MFColor         skyColor                [0 0 0]
        eventOut        SFBool          isBound
}
```

Узел Background применяется обычно в двух вариантах:

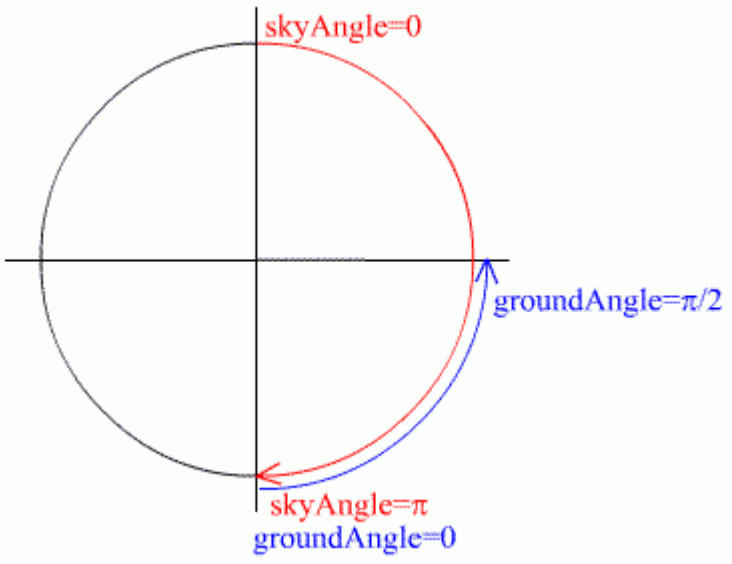
1) Приписывается цвет угловым интервалам на сфере бесконечного радиуса, и этим цветом заливаются концентрические сферические кольца.

```
Background {
        eventIn         SFBool          set_bind
        exposedField    MFFloat         groundAngle             []
        exposedField    MFColor         groundColor             []
        exposedField    MFFloat         skyAngle                []
        exposedField    MFColor         skyColor                [0 0 0]
        eventOut        SFBool          isBound
}
```

Назначение полей:

- set_bind – см. раздел 5.1.
- skyAngle – здесь перечисляются углы внутри сферы (их количество на 1 меньше, чем количество цветов в skyColor), определяющие сферические кольца. Диапазон изменения углов от 0 до Pi, 0 соответствует верхней точке сферы.
- skyColor – последовательно определяются цвета, в которые будут покрашены сферические кольца. Первый цвет соответствует верхней точке сферы, остальные цвета соответствуют кольцам, определенным в skyAngle. Цвета между кольцами линейно интерполируются (если в браузере MS VRML Viewer 2.1 включен режим High Color).
- groundAngle – то же, что и skyAngle, только диапазон изменения углов от 0 до Pi/2, где 0 соответствует нижней точке сферы.
- groundColor – то же, что и skyColor, но для groundAngle.
- isBound - см. раздел 5.1.

Для определения всех сферических колец  и соответствующих им цветов можно пользоваться только параметрами skyColor и skyAngle за счет диапазона изменения углов.




**Пример (файл 6-1-1.wrl):** иллюстрирует смену фона (сфера бесконечного радиуса) при нажатии на TouchSensor.

```
#VRML V2.0 utf8

Background
{
      skyAngle [1.57,3.14]
      skyColor [0 1 0,0 0 1,0 1 0]
}

DEF BK Background
{
      skyAngle [1.57,3.14]
      skyColor [1 0 0,0 1 0,1 0 0]
}

# красный куб, к которому прикреплен TouchSensor
Transform
{
  translation 0 3 5
  children [
      DEF TS TouchSensor
      {
      }
      Shape {
            appearance Appearance {
                  material Material {
                        diffuseColor 1 0 0
                  }
            }
            geometry Box {
            }
      }
  ]
}

# Камера
Viewpoint
{
      position 0 7.5 -3
      orientation 0.2 1 0.2 3.14
      description "Камера"
}
```

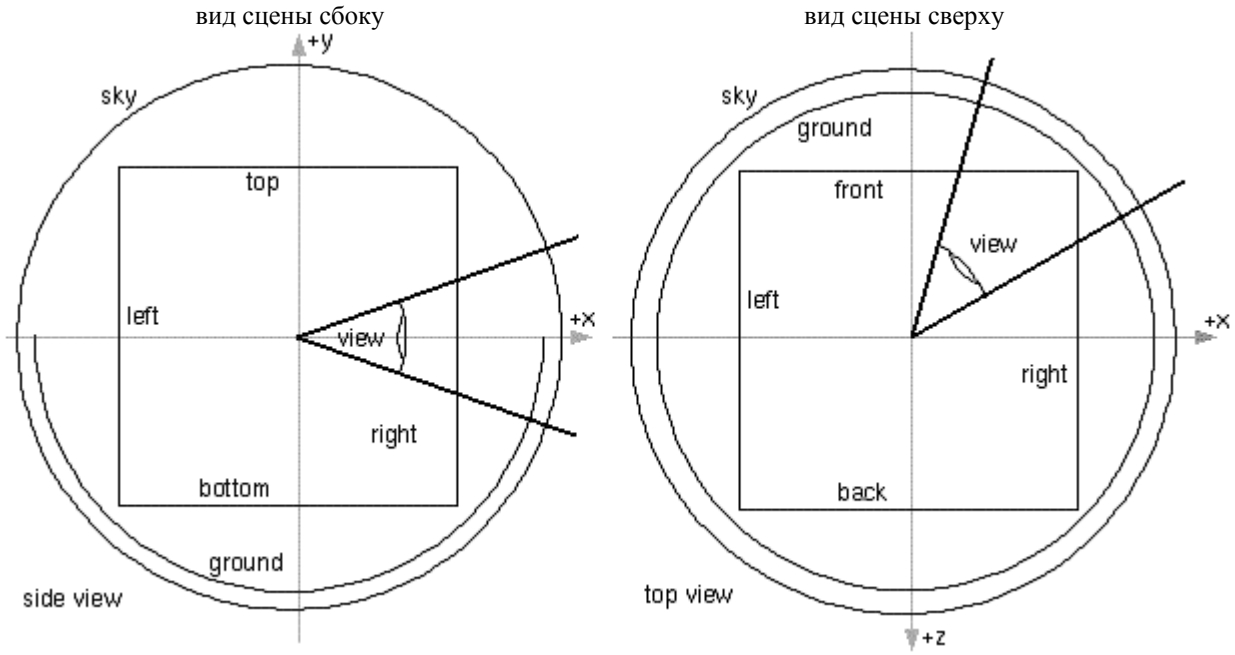
```
# по нажатию на TouchSensor BK перемещается в вершину
# связывающего стека
ROUTE TS.isActive TO BK.set_bind
```

2) Куб бесконечного объема, внутри которого находится пользователь, покрывается изнутри панорамными изображениями.

```
Background {
        eventIn         SFBool          set_bind
        exposedField    MFString        backUrl                 []
        exposedField    MFString        bottomUrl               []
        exposedField    MFString        frontUrl                []
        exposedField    MFString        leftUrl                 []
        exposedField    MFString        rightUrl                []
        exposedField    MFString        topUrl                  []
        eventOut        SFBool          isBound
}
```

Назначение полей:

- backUrl – название файла с изображением для задней стенки куба.
- bottomUrl – для нижней стенки.
- frontUrl – для передней.
- leftUrl – для левой.
- rightUrl – для правой.
- topUrl – для верхней.




пользователь находится в центре изображений

**Пример (файл 6-1-2.wrl):** здесь совмещены первый и второй способы применения узла BackGround (topUrl не указан, поэтому можно видеть верхнюю часть сферы).

```
#VRML V2.0 utf8

Background
{
        skyAngle [1.2]
        skyColor [0 0 1,1 1 1]
        backUrl "BACK.JPG"
        bottomUrl "FLOOR.JPG"
        frontUrl "BACK.JPG"
        leftUrl "BACK.JPG"
```

```
        rightUrl "BACK.JPG"

}

# красный куб
Transform
{
  translation 0 3 5
  children [
        Shape {
                appearance Appearance {
                        material Material {
                                diffuseColor 1 0 0
                        }
                }
                geometry Box {
                }
        }
  ]
}

# Камера
Viewpoint
{
        position 0 7.5 -3
        orientation 0.2 1 0.2 3.14
        description "Камера"
}
```

### 6.1.2. Fog (Туман).

Синтаксис узла:

```
Fog {
        eventIn         SFBool          set_bind
        exposedField    SFColor         color                   1 1 1
        exposedField    SFString        fogType                 "LINEAR"
        exposedField    SFFloat         visibilityRange         0
        eventOut        SFBool          isBound
}
```

Для создания эффекта реального тумана необходимо определять фон такого же цвета, как и туман.

Назначение полей:

- set_bind – см. раздел 5.1.
- color – цвет тумана.
- fogType – тип тумана - “LINEAR” (линейный) или “EXPONENTIAL” (экспоненциальный), но в MS VRML VIEWER поддерживается только тип “LINEAR”.
- visibilityRange – диапазон видимости в тумане (по умолчанию – туман на видимость не воздействует (тумана нет)).
- isBound - см. раздел 5.1.

**Пример (файл 6-2.wrl):** красный куб в сером тумане.

```
#VRML V2.0 utf8

Fog
{
      color 0.5 0.5 0.5
      visibilityRange 10
}

Background
{
```

```
      skyColor 0.5 0.5 0.5
}

# красный куб
Transform
{
  translation 0 3 5
  children [
      Shape {
            appearance Appearance {
                  material Material {
                        diffuseColor 1 0 0
                  }
            }
            geometry Box {
            }
      }
  ]
}

# Камера
Viewpoint
{
      position 0 7.5 -3
      orientation 0.2 1 0.2 3.14
      description "Камера"
}
```

### 6.1.3. Billboard (Модификатор осей координат).

Синтаксис узла:

```
Billboard {
        eventIn         MFNode          addChildren
        eventIn         MFNode          removeChildren
        exposedField    SFVec3f         axisOfRotation          0 1 0
        exposedField    MFNode          children                []
        field           SFVec3f         bboxCenter              0 0 0
        field           SFVec3f         bboxSize                -1 -1 -1
}
```

Все дочерние объекты, перечисленные в этом узле, ориентируются на точку наблюдения пользователя вокруг оси(осей), заданной в поле axisOfRotation.

Назначение полей:

- addChildren | - это входные события соответственно для добавления/удаления
- removeChildren | потомков.

- children – потомки.

- axisOfRotation – оси(ось) вращения дочерних объектов. По умолчанию объекты вращаются только вокруг оси Y. Для того, чтобы вращать объекты вокруг всех осей, значение этого параметра должно быть [0 0 0].

- bboxCenter | - эти 2 поля определяют центр и размеры параллелепипеда, в пределах
- bboxSize | которого существуют дочерние узлы. Это может использоваться с целью | оптимизации сцены. Результат не определен, если в размеры параллелепипеда | не влезает хотя бы один из потомков. Значение по умолчанию этих двух узлов | подразумевает, что параллелепипед не существует.

**Пример (файл 6-3.wrl):** две пальмы ориентируются на точку наблюдения пользователя – одна вокруг всех осей, другая только вокруг оси Y (по реализации пальма представляет из себя

параллелепипед маленькой толщины с прикрепленной к нему прозрачной текстурой в форме дерева).

```
#VRML V2.0 utf8

# небо и море
Background {
	skyColor 0.45 0.89 0.97
	groundAngle [1.5]
	groundColor [0 0 1, 0 0 1]
}

# остров
Shape {
	appearance Appearance {
		material Material {
			diffuseColor 1 1 0
		}
	}
	geometry Box {
		size 20 0.1 20
	}
}

# пальма №1
Transform {
	translation 2 2 -5
	children [
		Billboard {
			children [
				DEF PALM Shape {
					appearance Appearance {
						material Material {
							transparency 1
						}
						texture ImageTexture {
							url "tree.png"
						}
					}
					geometry Box {
						size 1 4 0.0001
					}
				}
			]
		}
	]
}

# пальма №2
Transform {
	translation 1 2 1
	children [
		Billboard {
			#вращаем вокруг всех осей
			axisOfRotation 0 0 0
			children [
				USE PALM
			]
		}
	]
}

Viewpoint {
	position -5 1.5 0
	orientation 0 1 0 -1.4
}
```

## 6.2. Звук.

### 6.2.1. AudioClip (Аудио-клип).

Синтаксис узла:

```
AudioClip {
        exposedField    SFString        description             ""
        exposedField    SFBool          loop                    FALSE
        exposedField    SFFloat         pitch                   1.0
        exposedField    SFTime          startTime               0
        exposedField    SFTime          stopTime                0
        exposedField    MFString        url                     []
        eventOut        SFTime          duration_changed
        eventOut        SFBool          isActive
}
```

Этот узел используется в качестве значения одного из полей узла Sound.

Назначение полей:

- description – текстовое описание источника звука.
- loop – звук зацикливается (при значении TRUE), или нет (при значении FALSE).
- pitch – скорость воспроизведения звука (например, 1.0 – нормальная скорость, 0.5 – вдвое медленнее оригинала, 2.0 – вдвое быстрее).
- startTime – время начала воспроизведения звукового файла.
- stopTime – время остановки воспроизведения.
- url – список звуковых файлов формата WAV или MIDI. Браузер проигрывает первый файл, который сможет загрузить.
- duration_changed – это событие генерируется в случае изменения длительности проигрываемого звука, но изменение значения pitch не влияет на это событие (оно генерируется, если, например, изменить источник звука). Если значение этого события = -1, это говорит о том, что звуковые данные еще не загрузились, или значение по каким-то причинам недоступно.
- isActive – если звук проигрывается, то генерируется TRUE. В противном случае генерируется FALSE.

**Пример:** в примере по узлу Sound.

### 6.2.2. Sound (Звук).

Синтаксис узла:

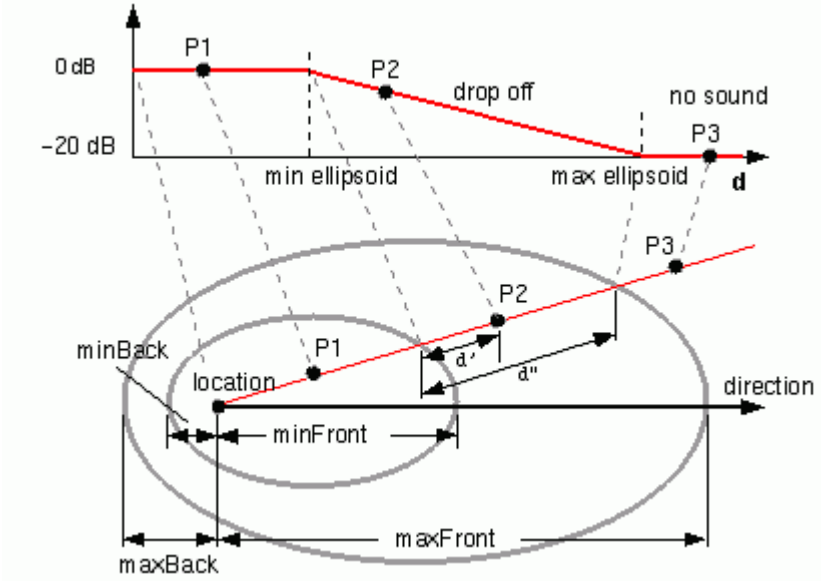
```
Sound {
        exposedField    SFVec3f         direction               0 0 1
        exposedField    SFFloat         intensity               1
        exposedField    SFVec3f         location                0 0 0
        exposedField    SFFloat         maxBack                 10
        exposedField    SFFloat         maxFront                10
        exposedField    SFFloat         minBack                 1
        exposedField    SFFloat         minFront                1
        exposedField    SFFloat         priority                0
        exposedField    SFNode          source                  NULL
        field           SFBool          spatialize              TRUE
}
```

Назначение полей:

- direction – вектор направления звука.
- intensity – громкость звука (от 0 до 1).
- location – местоположение источника звука в сцене.
- minBack | - в пределах сферы (см. рис. ниже), определенной этими параметрами,
- minFront | звук слышен с одинаковой громкостью, равной значению intensity.

- maxBack | - между сферой №2, определенной этими параметрами, и сферой №1,
- maxFront | определенной параметрами minBack и minFront, громкость звука
  | линейно интерполируется от значения 0 до intensity (см. рис. ниже).
- priority – приоритет звука (от 0 до 1). Высший приоритет связан со значением 1. Это поле введено в связи с тем, что браузер проигрывает ограниченное число звуковых файлов, и при превышении этого количества файлы с низким приоритетом проигрываться не будут.
- source – источник (в качестве значения здесь следует указывать узел AudioClip).
- spatialize – если значении TRUE, то создается эффект пространственного звука (при перемещении пользователя рядом с источником звука будет слышно перемещение звука).

Нижний рисунок – вид сцены сверху (сбоку).
Верхний – это график, который устанавливает соответствие между уровнем звука и положением пользователя в сцене.




**Пример (файл 6-4.wrl):** когда мы сталкиваемся со сферой, раздается звук.

```
#VRML V2.0 utf8

Background {
      skyAngle [3.14]
      skyColor [0 0 1, 1 1 1]
}


DEF COL Collision
{
      children
      [
            Shape {   # сфера
                  appearance Appearance {
                        material Material {
                              diffuseColor 1 1 0
                         }
                  }
                  geometry Sphere {
                  }
            }
      ]
}

Sound {
      minFront 5
      minBack 5
```

```
      source DEF SND AudioClip {
            url "tada.wav"
      }
}

# звук начинает проигрываться в то время, когда происходит столкновение
ROUTE COL.collideTime TO SND.startTime
```

# Тема 7.

## Скрипты и переключатели

### 7.1. Script (Скрипт, сценарий).

Полный синтаксис узла:

```
Script {
        exposedField    MFString        url                     []
        field           SFBool          directOutput            FALSE
        field           SFBool          mustEvaluate            FALSE

        а также любые из следующих полей:
        eventIn         Type            name
        field           Type            name                    default
        eventOut        Type            name
}
```

Этот узел позволяет создавать новые узлы со своим набором полей, входных и выходных событий.
Мы перечисляем набор полей, входных и выходных событий, которые мы бы хотели определить. Поле типа exposedField мы определить не можем.

Назначение полей:

- url – здесь определяется сценарий (программа).

Это делается одним из трех способов:

1) можно использовать откомпилированную программу на Java; в этом случае в поле url указывается название внешнего файла.

например:

```
url "abc.class"
```

2) можно использовать код, написанный на языке JavaScript из внешнего файла (с расширением .js).

например:

```
url "abc.js"
```

3) можно включать код, написанный на языке JavaScript, непосредственно в поле url.

например:

```
url "javascript:
      ...
      <программа>
      ...
      "
```

- mustEvalute – если значение этого поля = TRUE, то посланные узлу события обрабатываются в первую очередь. При значении этого поля равном FALSE, если появляется несколько событий одновременно, браузер строит все появившиеся события в очередь, и обрабатывает в той последовательности, которую сочтет наиболее эффективной.
- directOutput – если значение этого поля = TRUE, то браузер для данного узла игнорирует цепочку событий, и помещает значения непосредственно в поля тех узлов, которые доступны узлу Script.

При написании программы в поле URL основной задачей является связь входных событий узла Script с выходными. Эта программа состоит из функций, каждая из которых вызывается в определенный момент времени. Функции, соответствующие именам входных событий, определенных в узле, вызываются при появлении этих событий. Функция initialize() вызывается, как только завершается загрузка мира. Когда завершается работа со сценой, вызывается функция shutdown(). Функция eventProcessed() вызывается после обработки ряда

событий (как часто будет вызываться эта функция, определяется браузером). Чтобы послать исходящее событие из узла Script, нужно присвоить в программе значение для имени этого события.

Например:

```
Script {
      eventIn SFTime touchTime_f    # определяем
      eventIn SFTime touchTime_b    # 2 входящих события

      field SFInt32 number 10       # определяем переменную number
                                    # с начальным значением 10

      eventOut SFInt32 output       # определяем одно выходящее событие

      url "javascript:

            # эта функция вызывается сразу после завершения загрузки сцены
            function initialize() {
                  output = 0;
                  out_light=FALSE;
            }

            # эта функция вызывается при наступлении события touchTime_f
            # она генерирует значение выходного события output
            function touchTime_f(value, time) {
                  if (output == number) output = 0;
                  else ++output;
            }


            # эта функция вызывается при наступлении события touchTime_b
            # она генерирует значение выходного события output
            function touchTime_b(value, time) {
                  if (output == 0) output = number;
                  else --output;
            }"
}
```

**Пример:** в примере по узлу Switch.

### 7.2. Switch (Переключатель).

Синтаксис узла:

```
Switch {
        exposedField    MFNode          choice                  []
        exposedField    SFInt32         whichChoice             -1
}
```

Этот узел позволяет определить набор узлов и переключаться между ними. Активным является узел, выбираемый значением поля whichChoice. Неактивные узлы не генерируют событий и ничего в сцене не определяют.

Назначение полей:

- choice – набор узлов. Узлы, которые задаются в этом поле, должны относиться к узлам-потомкам (см. “структура VRML.doc”), т.е. в этом поле не может быть указан, например, узел Appearance.
- whichChoice – порядковый номер активного узла (считается, что нумерация начинается от 0, т.е. 0 – это первый узел, 1 – второй, и т.д.).

**Пример (файл 7-2.wrl):** создается SlideShow.

```
#VRML V2.0 utf8

PROTO PIC
[
      field MFString pic      ""
]

{
      Shape {
            appearance Appearance {
            texture ImageTexture {url IS pic}
            }
            geometry Box {size 100 80 2}
      }

}


PROTO STAND
[
      exposedField SFVec3f    Translation 0 0 0
      exposedField SFRotation Rotation 0 0 0 0
      exposedField SFVec3f    Scale 1 1 1
      field SFInt32     col   0
      field MFString    PIC0  "blank.jpg"
      field MFString    PIC1  "blank.jpg"
      field MFString    PIC2  "blank.jpg"
      field MFString    PIC3  "blank.jpg"
      field MFString    PIC4  "blank.jpg"
      field MFString    PIC5  "blank.jpg"
      field MFString    PIC6  "blank.jpg"
      field MFString    PIC7  "blank.jpg"
      field MFString    PIC8  "blank.jpg"
      field MFString    PIC9  "blank.jpg"

]
{
      Transform
      {
            translation IS Translation
            rotation IS Rotation
            scale IS Scale
            children
            [
#основа
                  Transform
                  {
                  translation 0 -6 -5.1
                  children
                  [
                        Shape  {
                        appearance Appearance {
                        material Material {diffuseColor 0.5 0.5 0.5}
                        }
                        geometry Box {size 120 100 10}
                        }
                  ]
                  }

#изображения
                  DEF SWITCH Switch {
                        whichChoice 0
                        choice [
                              PIC { pic IS PIC0}
                              PIC { pic IS PIC1}
```

```
                        PIC { pic IS PIC2}
                        PIC { pic IS PIC3}
                        PIC { pic IS PIC4}
                        PIC { pic IS PIC5}
                        PIC { pic IS PIC6}
                        PIC { pic IS PIC7}
                        PIC { pic IS PIC8}
                        PIC { pic IS PIC9}
                  ]
            }


#влево
            Group
            {
            children
            [
                  DEF sens_b TouchSensor {}
                  Transform
                  {
                  translation -40 -47 0
                  rotation 0 0 1 1.57
                  scale 1 1 0.1
                  children
                  [
                        Shape  {
                        appearance Appearance {
                        material Material {diffuseColor 0.8 0 0}
                        }
                        geometry Cone
                        {
                              bottomRadius 5
                              height 10
                        }
                        }
                  ]
                  }
            ]
            }

#вправо
            Group
            {
            children
            [
                  DEF sens_f TouchSensor {}
                  Transform
                  {
                  translation 40 -47 0
                  rotation 0 0 1 -1.57
                  scale 1 1 0.1
                  children
                  [
                        Shape  {
                        appearance Appearance {
                        material Material {diffuseColor 0.8 0 0}
                        }
                        geometry Cone
                        {
                              bottomRadius 5
                              height 10
                        }
                        }
                  ]
                  }
            ]
```

```
            }


            ]
      }


#SCRIPT
DEF CYCLE Script {
      eventIn SFTime touchTime_f
      eventIn SFTime touchTime_b
      field SFInt32 number IS col
      eventOut SFInt32 output
      url "javascript:
            function initialize() {
                  output = 0;
                  out_light=FALSE;
            }
            function touchTime_f(value, time) {
                  if (output == number) output = 0;
                  else ++output;
            }
            function touchTime_b(value, time) {
                  if (output == 0) output = number;
                  else --output;
            }"
}
ROUTE sens_f.touchTime TO CYCLE.touchTime_f
ROUTE sens_b.touchTime TO CYCLE.touchTime_b
ROUTE CYCLE.output TO SWITCH.whichChoice
}

#Основная программа
STAND {
#     Translation -0.22 4.095 11.75
#     Rotation 1 0 0 -0.21
      Scale 0.045 0.045 0.01
      col 9
      PIC0 "0.jpg"
      PIC1 "1.jpg"
      PIC2 "2.jpg"
      PIC3 "3.jpg"
      PIC4 "4.jpg"
      PIC5 "5.jpg"
      PIC6 "6.jpg"
      PIC7 "7.jpg"
      PIC8 "8.gif"
      PIC9 "9.gif"
}
```