Общие сведения

HTML5 – последний (на 2013 год незаконченный) стандарт языка HTML (HyperText Markup Language, язык разметки гипертекста), использующегося для создания веб-страниц.

Стандарт, как и предыдущие его версии, находится в разработке при участии W3C (World Wide Web Consortium, Консорциума Всемирной паутины) и является существенным улучшением стандарта HTML 4.01, стандартизованного комитетом ISO в 2000 году.

В частности, черновик стандарта предусматривает возможность включения в веб-страницу интерактивной 2D- и 3D-графики. Причем регламентируется возможность как императивного описания графической сцены (с использованием процедурного API на языке JavaScript), так и декларативного описания (с использованием графа сцены и включением описания сцены напрямую в HTML-страницу с интеграцией в DOM (Document Object Model, объектная модель документа)).

Для двухмерных интерактивных построений императивное средство реализации представлено элементом <canvas> (глава 4.8.11 черновика стандарта), а декларативное – элементом <svg>, использующим одноименный стандартизованный основанный на XML формат SVG (Scalable Vector Graphics, масштабируемая векторная графика) (глава 4.8.16 черновика стандарта).

Средства для отображения трехмерных сцен на 2013 год не прошли процедуру включения в черновик стандарта, но уже существуют их реализации. Так, для императивного отображения графики используется библиотека WebGL (Web-based Graphics Library, графическая библиотека для веба), разрабатываемого Khronos Group, а также компаниями Mozilla, Google, Apple, Opera, AMD и Nvidia. WebGL представляет собой контекст элемента <canvas>, обеспечивающий выполнение кода JavaScript напрямую на GPU с использованием аппаратного ускорения, без использования плагинов. Технология WebGL начала разрабатываться в 2006 году, а первая спецификация была выпущена в 2011 году.

Декларативное средство отображения 3D-графики представлено разрабатываемым институтом Fraunhofer проектом X3DOM. Проект начат в 2009 году.

X3DOM – JavaScript-библиотека, использующая WebGL для рендеринга внутри веб-страницы трехмерных сцен, описанных в формате X3D, без использования плагинов.

X3D – открытый, стандартизованный, основанный на XML (eXtensible Markup Language, расширяемый язык разметки) формат для представления интерактивных трехмерных сцен и объектов и обмена трехмерными данными между приложениями.

X3D был разработан консорциумом Web3D и стандартизован комитетом ISO. Последняя на данный момент версия стандарта 3.0 была принята в 2003 году. В настоящее время (конец 2013) осуществляется процесс стандартизации версии 3.3.

Формат X3D является эволюционным развитием формата VRML (Virtual Reality Modelling Language, язык моделирования виртуальной реальности), использовавшегося в 1990-х годах.

Граф сцены

Граф сцены – используемый в X3D способ задания трехмерных моделей, нацеленный на логическое и пространственное структурирование компонентов сцены. Помимо X3D метод графа сцены используется во многих редакторах (CorelDraw, AutoCAD) и графических API (OpenSceneGraph, OpenSG)

Граф сцены имеет древовидную структуру, которая иерархически определяет геометрию объектов сцены, их внешний вид, позицию в пространстве, ориентацию и т.д. Узлы графа представляют собой объекты, применяемые к ним преобразования, материалы, источники освещения и так далее. Узлы содержат количественные и качественные параметры, выраженные в значениях атрибутов. В случае если узел определяет преобразование, оно применяется ко всему подграфу (всем дочерним узлам). Среди узлов подграфа также могут быть узлы, определяющие преобразования. В этом случае произведение всех матриц преобразований от конкретного узла-объекта к корню графа даст матрицу преобразования из локальной системы координат в глобальную.

Граф сцены может динамически модифицироваться (узлы могут добавляться и удаляться, значения атрибутов узла могут изменяться) и реагировать на пользовательские действия. При визуализации браузер обходит дерево графа сцены, определяет изменившиеся узлы и визуализирует сцену с использованием доступных оптимизаций. Подобного рода оптимизации управляются из кода сцены. К ним относятся тиражирование подграфов сцены (DEF/USE), группировка, задание bounding box, включение backface culling, объявление convex-полигонов и другие. Их использование позволяет уменьшить вычислительную сложность рендеринга сцены, что критично для визуализации больших сцен в реальном времени. Подробнее различные виды оптимизаций будут рассмотрены далее.

Описание узлов

Полную информацию о формате X3D можно получить в постоянно пополняемых спецификациях стандарта X3D, доступных онлайн по адресу http://www.web3d.org/files/specifications/19775-1/V3.3/Part01/Architecture.html

При описании узла в спецификации указывается следующая информация:

1. Имя узла
2. Абстрактные типы узлов, от которых наследуется данный узел
3. Список спецификаций полей данного узла, каждый пункт которого включает в себя:
    a. Тип поля (SFFloat, MFInt32…)
    b. Тип доступа к полю ([], [in], [out], [in,out]). Подробнее типы доступа будут рассмотрены при обсуждении событий. При проектировании сцены могут указываться значения только для полей с типами доступа [] и [in,out]
    c. Имя поля
    d. Значение поля по умолчанию
    e. Допустимые значения поля

В X3D, который является словарем XML, узлы графа сцены описываются в виде элементов XML (парных или непарных тегов), а поля – в виде атрибутов узла (наборов `ключ="значение"` в открывающем теге элемента).

Например,

```
<Viewpoint orientation="0 -1 0 0.05" position="0.13 2.51 11.24"> </Viewpoint>
```

При задании значения атрибута могут использоваться как двойные, так и одинарные кавычки. При этом вид открывающей и закрывающей кавычек должен совпадать.

Между открывающим и закрывающим тегами узла могут быть определены дочерние узлы графа сцены.

Типы полей, используемые в X3D, приведены в Приложении 1.

Создание и просмотр сцены X3DOM

Сцена X3DOM представляет собой встроенный напрямую в код HTML-страницы элемент <X3D>, в котором размещен XML-код, описывающий граф сцены.

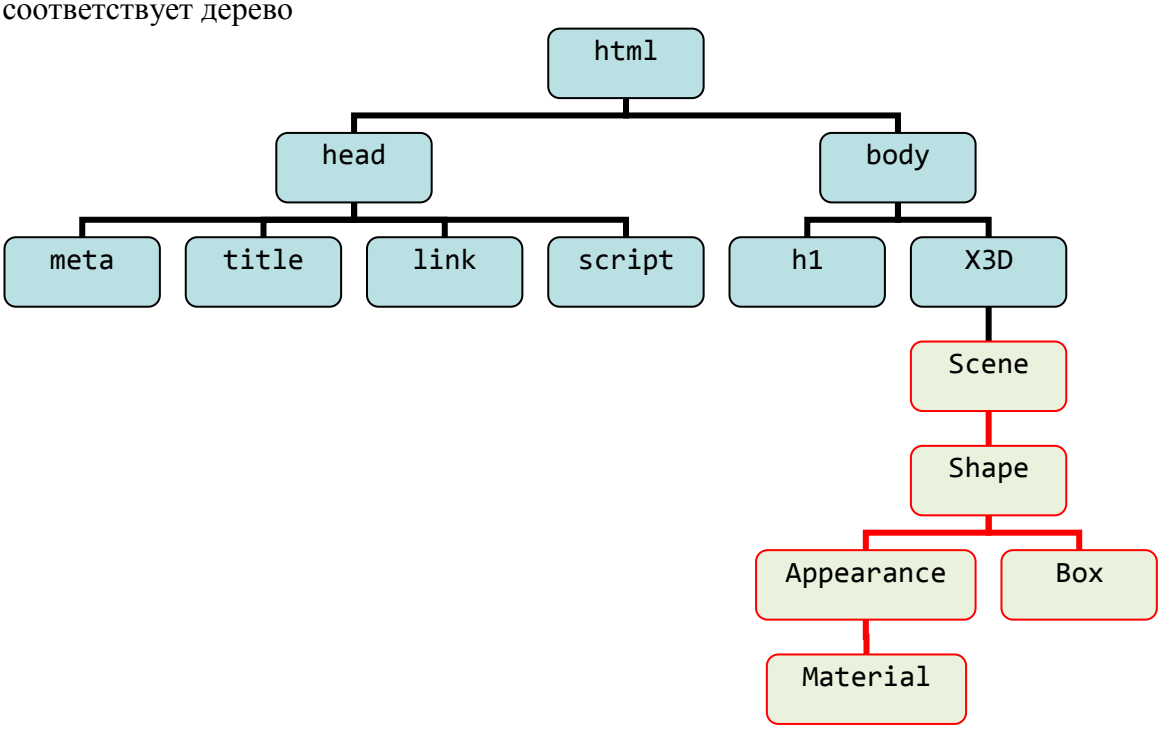
Граф сцены встроен в DOM.

DOM – древовидное представление HTML-документа. Все узлы документа образуют древовидную иерархическую структуру, при этом вложенные узлы являются дочерними по отношению к узлу, в который они вложены.

Так, документу

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8" />
    <title>Пример 1</title>
    <link rel="stylesheet" href="x3dom.css" />
    <script src="x3dom.js"></script>
  </head>
  <body>
    <h1>Пример 1</h1>
    <X3D>
      <Scene>
        <Shape>
          <Appearance>
            <Material></Material>
          </Appearance>
          <Box size="1 2 3"></Box>
        </Shape>
      </Scene>
    </X3D>
  </body>
</html>
```

соответствует дерево



В качестве атрибутов узла <X3D> могут быть указаны ряд параметров, влияющих на поведение библиотеки X3DOM. Полный список этих параметров указан в Приложении 2.

Внешний вид окна X3DOM можно гибко настроить, применяя к элементу <X3D> стили. Так, например, можно сделать окно полноэкранным:

```
<X3D style="left:0px; top:0px; width:100%; height:100%; border:none;">
```

В библиотеке X3DOM определены присутствующие в сцене по умолчанию методы навигации. Так, даже если определены только геометрические узлы, составляющие сцену, пользователь получает в свое распоряжение несколько предопределенных режимов навигации (ходьба, полет, осмотр и другие), а также горячих клавиш управления камерой. Описание предоставляемых библиотекой средств управления поведением сцены и навигации по ней приведено в Приложении 3.

Простые геометрические формы

Узел Shape является основополагающим в любой сцене и призван определять одиночный видимый объект.

```
Shape : X3DShapeNode {
  SFNode  [in,out] appearance NULL     [X3DAppearanceNode]
  SFNode  [in,out] geometry   NULL     [X3DGeometryNode]
  SFNode  [in,out] metadata   NULL     [X3DMetadataObject]
  SFVec3f []       bboxCenter 0 0 0    (-∞,∞)
  SFVec3f []       bboxSize   -1 -1 -1 [0,∞) or –1 –1 –1
}
```

Узел Shape в качестве дочернего может содержать один из возможных геометрических узлов (для задания геометрических свойств определяемого объекта). Также, он может содержать узел Appearance, отвечающий за внешний вид формы, который в свою очередь содержит узел Material, определяющий цветовые свойства объекта и/или узлы, отвечающие за текстурирование. Таким образом, Shape содержит совокупность информации о геометрических свойствах и свойствах внешнего вида для одиночного объекта сцены.

Простые геометрические формы описываются как один из заранее заданных геометрических примитивов, а детали отображения, в т. ч. триангуляция (разбиение на треугольники) осуществляется средствами WebGL.

У всех геометрических узлов есть одно общее важное свойство: solid.

Дело в том, что все треугольники (простейшие полигоны, из которых строится поверхность) имеют 2 стороны. При этом для повышения производительности широко применяется удаление невидимых граней (backface culling), когда рендерингу подвергается только одна из сторон полигона. По умолчанию удаление невидимых граней включено, что может вызвать неудобства при попадании «внутрь» геометрической формы, когда она становится невидимой для наблюдателя. Чтобы избежать такой ситуации, необходимо явно указывать в качестве значения параметра `solid="false"`, отключая таким образом удаление невидимых граней.

Единица измерения длины, принятая в X3D, – метр.

Все примитивы изначально располагаются в координатах (0,0,0).

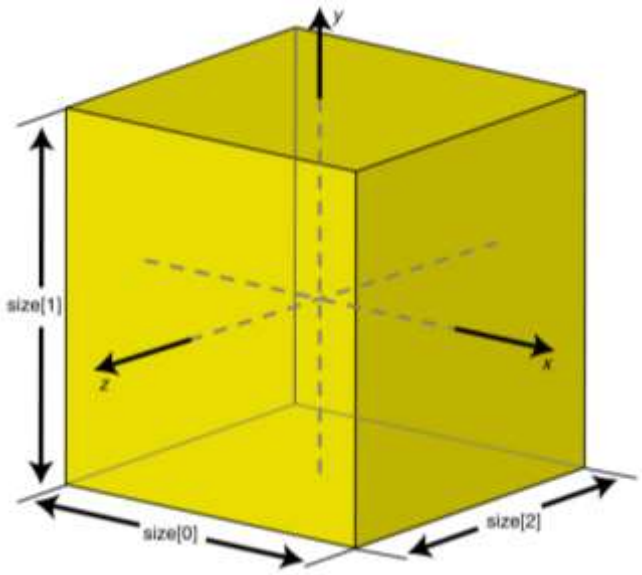
Еще одно общее свойство узлов – metadata – служит для добавления дополнительной информации (метаданных) к сцене.

Box

Определяет прямоугольный параллелепипед.

```
Box : X3DGeometryNode {
  SFVec3f []       size     2 2 2 (0,∞)
  SFBool  []       solid    true
}
```

Здесь и далее при описании будет опускаться поле metadata, присутствующее во всех узлах.



Описание полей:

size – размерности параллелепипеда по трем осям координат. Вещественные положительные числа.

Пример

Создается параллелепипед размером 1x2x3 метров с материалом по умолчанию.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Пример 1</title>
    <link rel="stylesheet" href="x3dom.css" />
    <script src="x3dom.js"></script>
  </head>
  <body>
    <h1>Пример 1</h1>
    <X3D>
      <Scene>
        <Shape>
          <Appearance>
            <Material></Material>
          </Appearance>
          <Box size="1 2 3"></Box>
        </Shape>
      </Scene>
    </X3D>
  </body>
```

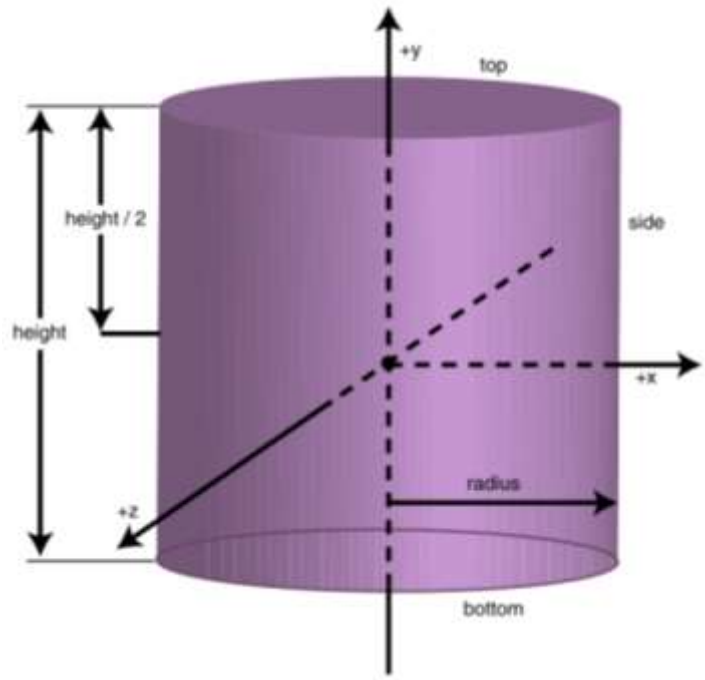
```
</html>
```

Данный пример содержит полный код html-документа со встроенной в него сценой. В дальнейшем для краткости в примерах будет приводиться только элемент <Scene>, отвечающий за содержимое сцены, а код страницы, включающей в себя X3D-элемент, будет подразумеваться.

Cylinder

Определяет цилиндр.

```
Cylinder : X3DGeometryNode {
  SFBool  []       bottom   true
  SFFloat []       height   2    (0,∞)
  SFFloat []       radius   1    (0,∞)
  SFBool  []       side     true
  SFBool  []       solid    true
  SFBool  []       top      true
}
```



Описание полей:
radius – радиус цилиндра.
height – высота цилиндра.
side – осуществляется ли рендеринг боковой поверхности цилиндра (true/false).
top – осуществляется ли рендеринг верхнего основания цилиндра (true/false).
bottom – осуществляется ли рендеринг нижнего основания цилиндра (true/false).

Если, к примеру, нижняя часть цилиндра перекрывается другим объектом, для ускорения рендеринга можно выставить параметр `bottom="false"`.

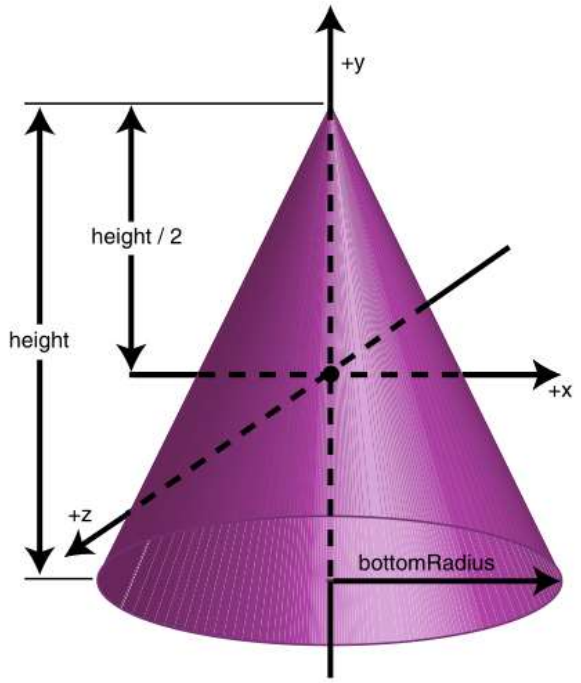
Cone

Определяет конус.

```
Cone : X3DGeometryNode {
  SFBool  []       bottom       true
  SFFloat []       bottomRadius 1    (0,∞)
  SFFloat []       height       2    (0,∞)
  SFBool  []       side         true
  SFBool  []       solid        true
}
```

Описание полей:
bottomRadius – радиус основания.
height – высота.
side – осуществляется ли рендеринг боковой поверхности конуса (true/false).
bottom – осуществляется ли рендеринг основания конуса (true/false).

По умолчанию конус располагается на 1 метр выше и ниже оси Y, и имеет радиус основания 1 метр. Его центральная ось параллельна оси Y.

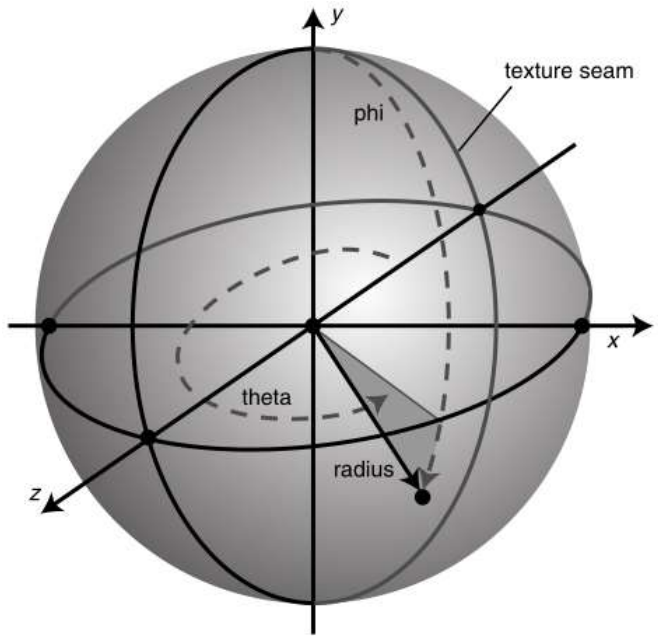


Sphere

Определяет сферу.

```
Sphere : X3DGeometryNode {
  SFFloat []       radius   1    (0,∞)
  SFBool  []       solid    true
}
```

Описание полей:
radius – радиус сферы



Text

Определяет 2D-текст.

```
Text : X3DGeometryNode {
  SFNode   [in,out] fontStyle  NULL  [X3FontStyleNode]
  MFFloat  [in,out] length     []    [0,∞)
  SFFloat  [in,out] maxExtent  0.0   [0,∞)
  MFString [in,out] string     []
  SFBool   []       solid      false
}
```

Описание полей узла Text:

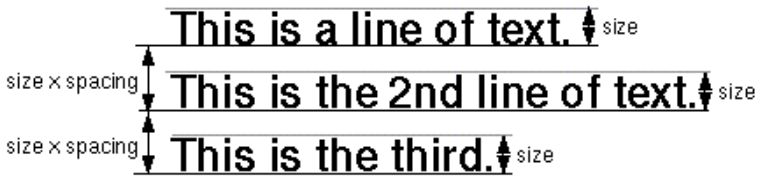
- string –строки текста, которые будут отображены.
- maxExtent - максимальная ширина текста (в метрах). Если самая длинная строка текста выходит за пределы этой ширины, весть текст будет сжат.
- length – массив чисел, определяющий ширину каждой из строк. Если ширина строки не совпадает с заданной соответствующим элементом массива, она будет сжата или растянута, чтобы ей соответствовать.

В узле Text в качестве дочернего используется узел FontStyle, определяющий параметры стиля текста.

```
FontStyle : X3DFontStyleNode {
  MFString []     family      "SERIF"
  SFBool   []     horizontal  true
  MFString []     justify     "BEGIN" ["BEGIN","END","FIRST","MIDDLE",""]
  SFString []     language    ""
  SFBool   []     leftToRight true
  SFFloat  []     size        1.0     (0,∞)
  SFFloat  []     spacing     1.0     [0,∞)
  SFString []     style       "PLAIN" ["PLAIN"|"BOLD"|"ITALIC"|"BOLDITALIC"|""]
  SFBool   []     topToBottom true
}
```

Описание полей узла FontStyle:

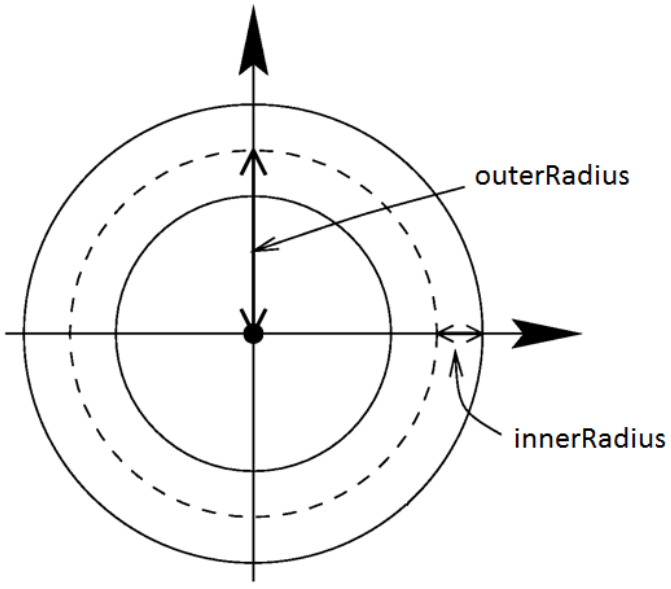
- size - высота строки текста (влияет также на межстрочный интервал).
- family – определяет последовательность гарнитур в порядке их предпочтения, например `["Lucida Sans Typewriter", "Lucida Sans", "Helvetica", "SANS"]`. Все браузеры по умолчанию поддерживают как минимум базовый набор из трех типов гарнитур, задаваемых ключевыми словами `"SERIF"` (пропорциональная с засечками), `"SANS"` (пропорциональная без засечек) и `"TYPEWRITER"` (моноширинная)
- style - стиль шрифта: `"PLAIN"` (обычный), `"BOLD"` (полужирный), `"ITALIC"` (курсив) или `"BOLDITALIC"` (полужирный курсив).
- 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-нормальное, 2-двойное, и т.д.).



Torus

Определяет тор.

Спецификация узла недоступна, поскольку узел является специфичным для X3DOM и не включен в стандарт X3D.



Описание полей:
innerRadius – малый радиус
outerRadius – большой радиус

Трансформация объектов

Transform

Используется для вращения, масштабирования и сдвига объектов.

```
Transform : X3DGroupingNode {
  MFNode     [in]     addChildren               [X3DChildNode]
  MFNode     [in]     removeChildren            [X3DChildNode]
  SFVec3f    [in,out] center           0 0 0    (-∞,∞)
  MFNode     [in,out] children         []       [X3DChildNode]
  SFRotation [in,out] rotation         0 0 1 0  [-1,1] or (-∞,∞)
  SFVec3f    [in,out] scale            1 1 1    (0,∞)
  SFRotation [in,out] scaleOrientation 0 0 1 0  [-1,1] or (-∞,∞)
  SFVec3f    [in,out] translation      0 0 0    (-∞,∞)
  SFVec3f    []       bboxCenter       0 0 0    (-∞,∞)
  SFVec3f    []       bboxSize         -1 -1 -1 [0,∞) or -1 -1 -1
}
```

Описание полей:

- center – определяет центр системы координат относительно оригинальной (0,0,0), в которой будет происходить масштабирование и вращение.
- children – дочерние узлы, подлежащие трансформации.
- rotation – указывается вектор, соответствующий оси вращения (3 числа – X Y Z), далее угол в радианах, на который должно быть осуществлено вращение относительно заданной оси. Например, `rotation="1 0 0 -0.707"` вращает объект или группу объектов относительно оси X на -0.707 радиан (-45˚).
- scale – расширение/сжатие (коэффициенты масштабирования по осям X Y Z).
- scaleOrientation – определяет временное вращение локальной системы координат перед масштабированием.
- translation – смещение по осям X Y Z.
- bboxCenter – центр bounding box.
- bboxSize – размеры bounding box.

Трансформация выполняется единообразно для всех объектов из children, объединяя их в единую группу.

Bounding box является невидимой замкнутой геометрической областью, которая содержит объекты трансформации и служит для оптимизации работы браузера, подсказывая ему размеры группы объектов с целью более эффективного выполнения операций отсечения. Задавать параметры bounding box необязательно.

Преобразования применяются к объекту (группе объектов) в такой последовательности:

1) Обратное смещение center для совмещения заданного центра временной локальной системы координат масштабирования и вращения с текущим центром оригинальной системы координат.
2) Обратное ориентирование scaleOrientation для совмещения осей координат временной локальной системы координат масштабирования с текущими осями оригинальной системы координат.
3) Масштабирование scale.
4) Прямое ориентирование scaleOrientation для восстановления исходной ориентации осей локальной системы координат.
5) Вращение rotation относительно произвольной оси для установления новой ориентации осей локальной системы координат.

6) Прямое смещение center для восстановления исходного центра локальной системы координат.
7) Смещение translation по осям глобальной системы координат для позиционирования нового центра локальной системы координат.

Если задана точка P в трехмерном пространстве и над ней задан узел Transform, P преобразуется в P' за 7 промежуточных шагов, которые могут быть выражены в виде умножения матриц соответствующих преобразований:

P' = T * C * R * SR * S * −SR * −C * P,
где T – матрица смещения translation,
C – матрица смещения center,
R – матрица вращения rotation,
SR – матрица вращения scaleOrientation,
S – матрица масштабирования scale.

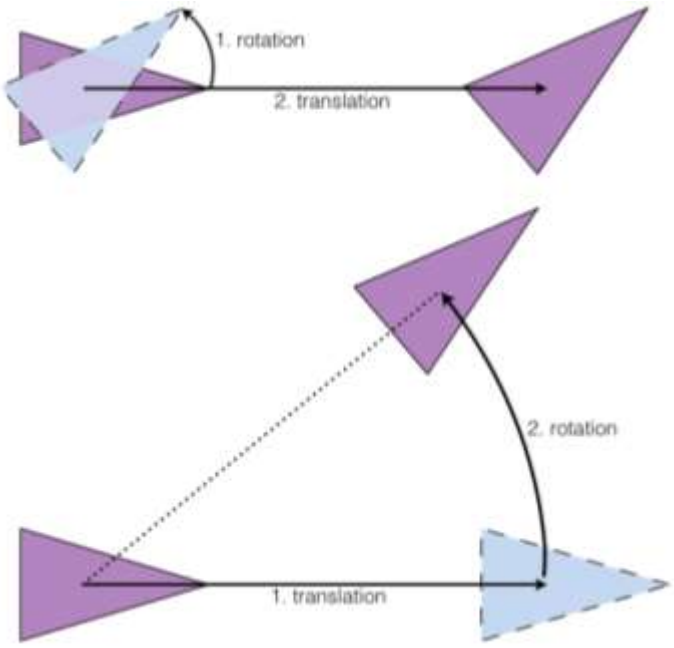
Иными словами, узел

```
<Transform center=C rotation=R scale=S scaleOrientation=SR translation=T>
  <…>
</Transform>
```

может быть эквивалентно преобразован в следующий набор вложенных (последовательно применяющихся) узлов:

```
<Transform translation=T>
  <Transform translation=C>
    <Transform rotation=R>
      <Transform rotation=SR>
        <Transform scale=S>
          <Transform rotation=-SR>
            <Transform translation=-C>
              <…>
            </Transform>
          </Transform>
        </Transform>
      </Transform>
    </Transform>
  </Transform>
</Transform>
```

Порядок применения операций трансформации важен, поскольку его изменение может привести к различному позиционированию объекта в пространстве. Например, вот как может отразиться на объекте замена метами операций translation и rotation.



Это обусловлено тем, что и смещение, и вращение происходит в глобальной системе координат, которая остается неизменной после выполнения первой операции.

Когда происходит трансформация, определяется новая локальная система координат, в которой располагаются все дочерние объекты. Таким образом, translation задает величину смещения центра локальной системы координат относительно центра глобальной, rotation – относительную ориентацию, а scaling – относительное масштабирование.

Для определения направления смещения, а также ориентирования оси вращения полезно знать направления осей координат. Для этого можно воспользоваться правилом правой руки. Если отогнутый большой палец правой указывает положительное направление оси X, вытянутый указательный – оси Y, то полусогнутый средний – оси Z.

Рисунок XXX – Правило правой руки для определения направления осей координат

Для определения положительного направления вращения также действует правило правой руки, но в ином виде. Если отогнутый большой палец правой указывает положительное направление оси вращения, то 4 согнутых пальца покажут положительное направление вращения относительно этой оси.

Рисунок XXX – Правило правой руки для определения положительного направления вращения

Внешний вид

Appearance

Описывет внешний вид объекта.

Является дочерним узлом Shape и предназначен для вмещения в себе информации о материале объекта (узлы Material, TwoSidedMaterial), его текстуре (узлы ImageTexture, MovieTexture, PixelTexture) и трансформации текстуры (узел TextureTransform).

```
Appearance : X3DAppearanceNode {
  SFNode [in,out] fillProperties   NULL [FillProperties]
  SFNode [in,out] lineProperties   NULL [LineProperties]
  SFNode [in,out] material         NULL [X3DMaterialNode]
  MFNode [in,out] shaders          []   [X3DShaderNode]
  SFNode [in,out] texture          NULL [X3DTextureNode]
  SFNode [in,out] textureTransform NULL [X3DTextureTransformNode]
}
```

Material

Описывает свойства материала объекта. Свойства материала применяются единообразно ко всем полигонам геометрического объекта. Помимо этого, материал определяет, как освещение сцены взаимодействует с объектом.

```
Material : X3DMaterialNode {
  SFFloat [in,out] ambientIntensity 0.2         [0,1]
  SFColor [in,out] diffuseColor     0.8 0.8 0.8 [0,1]
  SFColor [in,out] emissiveColor    0 0 0       [0,1]
  SFFloat [in,out] shininess        0.2         [0,1]
  SFColor [in,out] specularColor    0 0 0       [0,1]
  SFFloat [in,out] transparency     0           [0,1]
}
```

Описание 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). Описывает цвет отражаемого объектом света в зависимости от удаленности источника освещения и угла падения света на поверхность объекта.
- specularColor – цвет блеска освещённых участков объекта (R G B). Определяет цвет, который добавляется к отражаемому поверхностью объекта освещению, когда угол падения света примерно равен углу наблюдения.
- emissiveColor – какой цвет излучает объект (свет не отбрасывается на окружающие объекты) (R G B). Объекты светятся этим цветом даже если на них не направлено ни одного источника освещения.
- ambientIntensity – интенсивность отражаемого объектом рассеянного света (значение от 0 до 1). Рассеянный свет является всенаправленным и не зависит от расположения источников света, учитывается только их количество в сцене.
- shininess – определяет степень резкости освещения блестящих участков объекта (значение от 0 до 1). Блеск возникает, когда свет на поверхность падает под углом, близким к углу, под которым на поверхность смотрит наблюдатель. Низкие значения параметра дают более мягкий блеск, высокие – более жесткий.
- transparency – прозрачность объекта (значение от 0 до 1).

TwoSidedMaterial

Описывает двусторонний материал. Может применяться для полигонов, у которых отрисовываются обе стороны (solid="false").

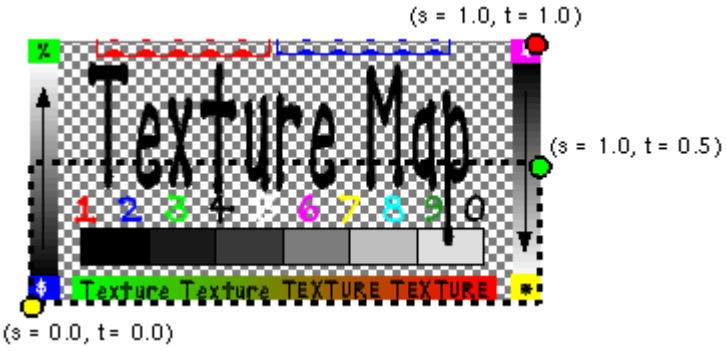
Содержит пары свойств материала – для лицевой стороны поверхности и для обратной.

```
TwoSidedMaterial : X3DMaterialNode {
  SFFloat [in,out] ambientIntensity     0.2         [0,1]
  SFFloat [in,out] backAmbientIntensity 0.2         [0,1]
  SFColor [in,out] backDiffuseColor     0.8 0.8 0.8 [0,1]
  SFColor [in,out] backEmissiveColor    0 0 0       [0,1]
  SFFloat [in,out] backShininess        0.2         [0,1]
  SFColor [in,out] backSpecularColor    0 0 0       [0,1]
  SFFloat [in,out] backTransparency     0           [0,1]
  SFColor [in,out] diffuseColor         0.8 0.8 0.8 [0,1]
  SFColor [in,out] emissiveColor        0 0 0       [0,1]
  SFFloat [in,out] shininess            0.2         [0,1]
  SFBool  [in,out] separateBackColor    false
  SFColor [in,out] specularColor        0 0 0       [0,1]
  SFFloat [in,out] transparency         0           [0,1]
}
```

Текстурирование

Текстурирование – наложение двумерного изображения на поверхность трехмерного объекта.

Текстуре присваивается локальная система координат (s,t) с диапазоном координат [0.0,1.0]. Ось s соответствует нижнему краю картинки, ось t - левому. Левый нижний пиксель изображения имеет координаты s=0.0, t=0.0, правый верхний – s=1.0, t=1.0.



Координаты текстур используются для контролирования замощения объекта текстурой по горизонтали и вертикали заданием параметров repeatS и repeatT, а также при преобразованиях текстуры (масштабировании, вращении), определяемых узлом TextureTransform.

ImageTexture

Задает для объекта текстуру из подгружаемого файла изображения. Поддерживаются форматы PNG, JPEG, GIF и другие.

```
ImageTexture : X3DTexture2DNode, X3DUrlObject {
  MFString [in,out] url              []   [URI]
  SFBool   []       repeatS          true
  SFBool   []       repeatT          true
  SFNode   []       textureProperties NULL [TextureProperties]
}
```

Описание полей:

url – набор путей к файлу. Рекомендуется для надежности указывать несколько путей, т.к. по некоторым из них файл текстуры может оказаться недоступен, что приведет к тому, что объект не будет текстурирован.

repeatS – использовать ли горизонтальное замощение поверхности текстурой вдоль оси s (true/false).

repeatT – использовать ли вертикальное замощение поверхности текстурой вдоль оси t (true/false).

MovieTexture

Используется для задания движущейся текстуры

```
MovieTexture : X3DTexture2DNode, X3DSoundSourceNode, X3DUrlObject {
  SFString [in,out] description       ""
  SFBool   [in,out] loop              FALSE
  SFTime   [in,out] pauseTime         0     (-∞,∞)
  SFFloat  [in,out] pitch             1.0   (0,∞)
  SFTime   [in,out] resumeTime        0     (-∞,∞)
  SFFloat  [in,out] speed             1.0   (-∞,∞)
  SFTime   [in,out] startTime         0     (-∞,∞)
  SFTime   [in,out] stopTime          0     (-∞,∞)
  MFString [in,out] url               []    [URI]
  SFTime   [out]    duration_changed
  SFTime   [out]    elapsedTime
  SFBool   [out]    isActive
  SFBool   [out]    isPaused
  SFBool   []       repeatS           TRUE
  SFBool   []       repeatT           TRUE
  SFNode   []       textureProperties NULL [TextureProperties]
}
```

PixelTexture

Используется для задания пиксельной текстуры.

```
PixelTexture : X3DTexture2DNode {
  SFImage [in,out] image             0 0 0
  SFBool  []       repeatS           TRUE
  SFBool  []       repeatT           TRUE
  SFNode  []       textureProperties NULL  [TextureProperties]
}
```

TextureTransform

Позволяет определить трансформацию текстуры – двумерные преобразования координат текстуры (двумерное смещение, вращение, масштабирование) для изменения способа ее нанесения на поверхность объекта. Преобразования применяются к системе координат (s,t), а не к самой текстуре, поэтому создается впечатление, что к самой текстуре применяются обратные преобразования.

Порядок применения преобразований: смещение, вращение (относительно произвольной точки в системе координат (s,t)), масштабирование.

```
TextureTransform : X3DTextureTransform2DNode {
  SFVec2f [in,out] center      0 0  (-∞,∞)
  SFFloat [in,out] rotation    0    (-∞,∞)
  SFVec2f [in,out] scale       1 1  (-∞,∞)
  SFVec2f [in,out] translation 0 0  (-∞,∞)
}
```

Описание полей:

translation – определяет смещение центра координат локальной системы координат текстуры (s,t) по поверхности объекта. Задается в локальной системе координат, представляет собой пару числе в интервале [0.0,1.0], соответствующих значениям смещения по s и по t.

rotation – определяет угол (в радианах) поворота осей координат (s,t).

center – определяет смещение, относительно которого будет осуществляться поворот.

scale – определяет коэффициенты масштабирования осей s и t.

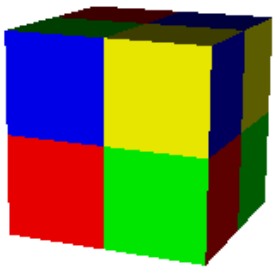


Рисунок XXX – Текстура без применения трансформаций

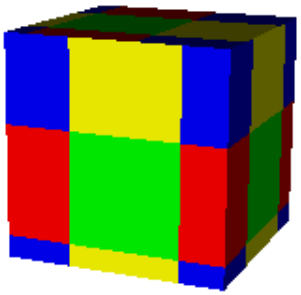


Рисунок XXX – С параметром `translation="0.2 -0.1"`

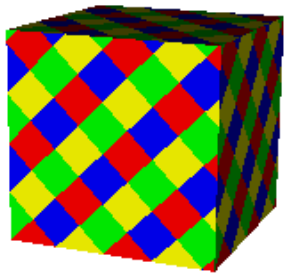


Рисунок XXX – С параметрами `scale="3 3" rotation="0.78"`

Поля узла TextureTransform могут быть анимированы для создания движущейся текстуры.

Сложные объекты

В X3D существуют геометрические узлы, позволяющие создавать объекты произвольной формы, не ограниченной набором заданных примитивов. Так же, как и узлы геометрических примитивов, они являются дочерними узлами узла Shape и схожим образом могут быть использованы в паре с узлом внешнего вида (Appearance), определяющим материал или текстурирование. Тем не менее, для сложных объектов существует возможность более детального задания параметров внешнего вида для отдельных поверхностей/полигонов/линий/точек с использованием узлов Color, ColorRGBA и Normal.

Сложные объекты описываются на низком уровне – путем задания массива координат вершин полигонов (вертексов) с последующим их индексированием и объединением в поверхности.

На каркасном представлении можно наглядно увидеть, что сложные объекты можно представить поверхностью, состоящей из плоских соприкасающихся друг с другом полигонов.

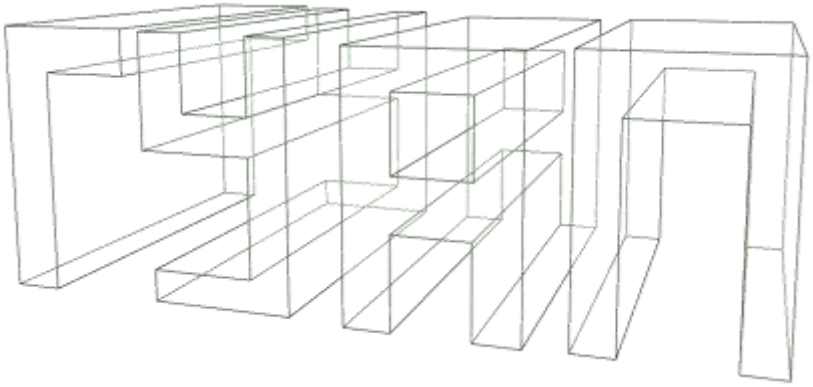


Рисунок XXX – Каркасное представление модели сцены

Все объекты в процессе рендеринга подвергаются триангуляции – разбиению поверхности на треугольники.

Основное правило при задании сложных объектов путем перечисления координат вершин их полигонов – избегать неплоских полигонов. Рекомендуется, чтобы все точки, образующие полигон, были компланарны, т.е. лежали в одной плоскости. Триангуляция неплоских полигонов может быть неоднозначной, что может вызвать непредсказуемые эффекты при рендеринге. В то же время проверка на плоскость полигонов не производится из соображений эффективности, и ответственность ложится целиком на разработчика.

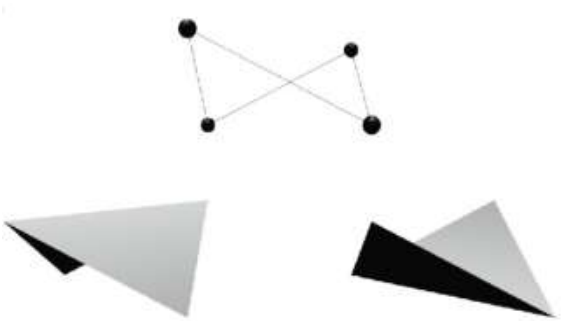


Рисунок XXX – Неоднозначность рендеринга неплоского полигона.

Перед рассмотрением узлов, отвечающих за создание непосредственно сложных объектов, целесообразно ознакомиться со служебными узлами, используемыми в них и задающими координаты, цвета и нормали.

Coordinate

Определяет массив координат в трехмерном пространстве. В последующем координаты могут быть проиндексированы для создания линий или полигонов сложных объектов.

```
Coordinate : X3DCoordinateNode {
  MFVec3f [in,out] point    []     (-∞,∞)
}
```

Описание полей:
point – массив координат.

Пример:

```
<Coordinate point="0 0 0, 1 1 1"></Coordinate>
```

Color и ColorRGBA

Color определяет массив цветов в пространстве RGB. Каждый элемент массива состоит из трех компонентов, представленных вещественными числами в интервале [0..1].

ColorRGBA определяет массив цветов в пространстве RGBA. Четвертый компонент – непрозрачность (альфа-канал). Также задается числом в интервале [0..1], где 0 – полная прозрачность, 1 – полная непрозрачность.

```
Color : X3DColorNode {
  MFColor [in,out] color    [NULL] [0,1]
}

ColorRGBA : X3DColorNode {
  MFColorRGBA [in,out] color    [NULL] [0,1]
}
```

Описание полей:
color – массив цветов

Примеры:

```
<Color color="0 0 0, 1 1 1, 1 0 0, 0 1 0, 0 0 1">
</Color>

<ColorRGBA color="0 0 0 0.5, 1 1 1 0.5, 1 0 0 0.5, 0 1 0 0.5, 0 0 1 0.5">
</ColorRGBA>
```

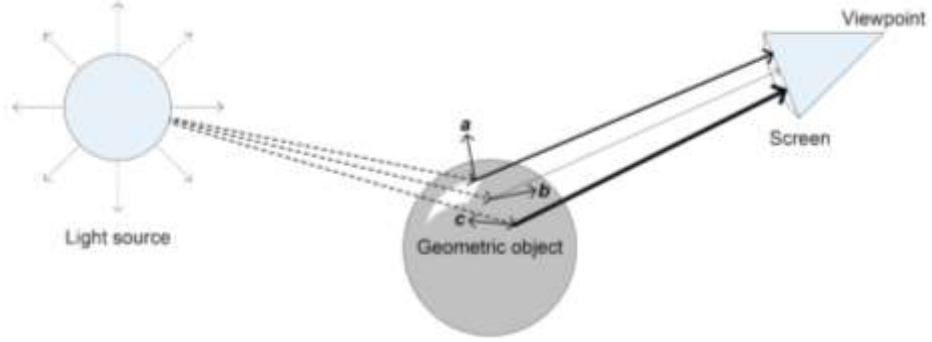
Normal

Определяет список векторов нормалей.

Нормаль – вектор, перпендикулярный полигону и используемый для расчета освещения объекта. Как правило, нормали вычисляются автоматически по координатам вертексов, образующих полигон. В X3D есть возможность ручного задания векторов нормалей для каждого полигона, что может преследовать две цели:

1) Повышение эффективности рендеринга за счет отсутствия необходимости вычисления векторов нормалей.

2) Применение специальных эффектов, основанных на освещении (возможно, связанных с анимацией векторов нормалей).



На приведенном рисунке три расположения векторов нормали влияют на освещенность поверхности следующим образом:

- вектор нормали a перпендикулярен объекту, освещенность соответствует естественной;
- вектор нормали b практически направлен в противоположную сторону от источника освещения, освещеность ниже естественной;
- вектор нормали c практически направлен в сторону источника освещения, освещеность выше естественной.

Вектор нормали рекомендуется нормализовать (преобразовать в вектор единичной длины). Этого можно достичь следующим образом:

$$x' = \frac{x}{\sqrt{x^2 + y^2 + z^2}};\ y' = \frac{y}{\sqrt{x^2 + y^2 + z^2}};\ z' = \frac{z}{\sqrt{x^2 + y^2 + z^2}}$$

$$N' = (x', y', z')$$

```
Normal : X3DNormalNode {
  MFVec3f [in,out] vector   []   [-1,1]
}
```

Описание полей:
vector – массив векторов нормалей

Пример.
Определяются три не нормализованных вектора:

```
<Normal vector="1 0 1, -1 0.5 -1, -2 0 -3"> </Normal>
```

После нормализации:

```
<Normal vector=".71 0 .71, -.67 .33 –.67, -.55 0 -.83"> </Normal>
```

Общие поля сложных геометрических узлов

ccw (counterclockwise, против часовой стрелки) определяет порядок описания вертексов для полигонов. По умолчанию имеет значение true, что соответствует правилу правой руки. Если четыре пальца правой руки указывают порядок описания вертексов, то отогнутый большой палец указывает направление вектора нормали лицевой стороны данного полигона.

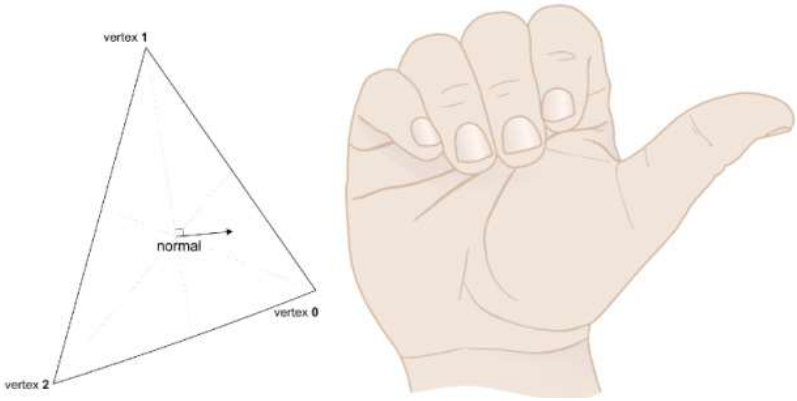


Рисунок XXX – Правило правой руки для определения лицевой стороны полигона

colorPerVertex определяет, применяются ли цвета к вертексам (true) или к полигонам/линиям (false). В первом случае для интервалов между вертексами осуществляется линейная (в случае закрашивания полигона – билинейная) интерполяция цвета, в массиве colorIndex количество цветов должно соответствовать количеству вертексов. Во втором случае цвета не интерполируются и применяются к полигонам/линиям, в массиве colorIndex количество цветов должно соответствовать количеству полигонов/линий.

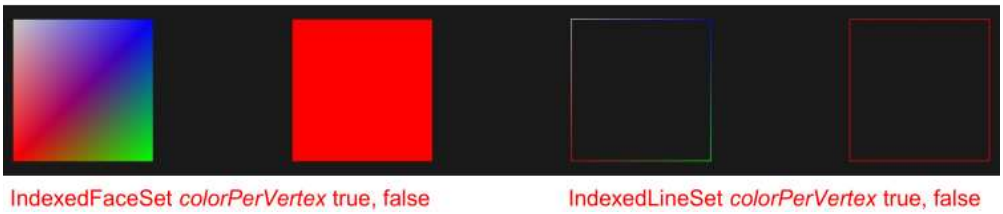


Рисунок XXX – Влияние параметра colorPerVertex на закрашивание объектов

convex устанавливает, являются ли все полигоны объекта выпуклыми (т.е. плоскими, не пересекающими самих себя и не имеющими внутренних углов, превышающих 180 градусов). Если указать в качестве значения true, рендеринг осуществляется эффективнее, поскольку отсутствуют вычислительные затраты на обработку невыпуклых полигонов.

creaseAngle определяет предельный угол (в радианах) между нормалями двух полигонов, когда для них должно осуществляться гладкое затенение. Если угол между нормалями больше, чем указанный в данном поле, используется негладкое затенение.

colorIndex, coordIndex, normalIndex - массивы, определяющие порядок применения значений массивов цветов, индексов и нормалей дочерних узлов Color (или ColorRGBA), Coordinate и Normal соответственно. Нумерация элементов массивов начинается с 0.

PointSet

Создает набор несвязанных между собой точек. Точки не участвуют в проверке на столкновения.

В качестве дочерних включает узлы Color и Coordinate. Значения массива узла Coordinate последовательно задают все точки, а значения массива узла Color – соответствующие им цвета.

Индексирование координат и цветов не производится, координаты просто выводятся в порядке их перечисление в массиве в дочернем узле Coordinate и окрашиваются согласно соответствующему элементу массива дочернего узла Color.

```
PointSet : X3DGeometryNode {
  MFNode [in,out] attrib   []   [X3DVertexAttributeNode]
  SFNode [in,out] color    NULL [X3DColorNode]
  SFNode [in,out] coord    NULL [X3DCoordinateNode]
  SFNode [in,out] fogCoord NULL [FogCoordinate]
}
```

Узел не имеет полей.

```
<PointSet>
    <Color color="1 0 0, 0 1 0, 0 0 1, 0.8 0.8 0.8"></Color>
    <Coordinate point="-2 0 0, 0 0 0, 0 0 2, 0 0 4"></Coordinate>
</PointSet>
```

Точки не имеют цвета поверхности, вместо этого они имеют излучаемый цвет. Поэтому вместо задания узла Color, можно раскрасить все точки одинаково с использованием поля emissiveColor узла Material.

IndexedLineSet

Позволяет определить набор ломаных линий определённого цвета в пространстве. Линии не участвуют в проверке на столкновения.

В качестве дочерних включает узлы Coordinate и Color, которые содержат координаты точек и цвета в произвольном порядке. Для определения порядка построения ломаных линий из элементов массива и их раскраски используются поля coordIndex и colorIndex.

```
IndexedLineSet : X3DGeometryNode {
  MFInt32 [in]     set_colorIndex
  MFInt32 [in]     set_coordIndex
  MFNode  [in,out] attrib         []   [X3DVertexAttributeNode]
  SFNode  [in,out] color          NULL [X3DColorNode]
  SFNode  [in,out] coord          NULL [X3DCoordinateNode]
  SFNode  [in,out] fogCoord       NULL [FogCoordinate]
  MFInt32 []       colorIndex     []   [0,∞) or -1
  SFBool  []       colorPerVertex TRUE
  MFInt32 []       coordIndex     []   [0,∞) or -1
}
```

Описание полей:

coordIndex определяет последовательность обхода координат для построения ломаных линий. Нумерация координат начинается с 0. Если описание одной из ломаных линий закончено, ставится маркер окончания ломаной линии (–1).

colorIndex определяет последовательность применения значений элементов массива цветов к точкам или линиям.

colorPerVertex определяет, применяются ли цвета к линиям (false) или к точкам, интерполируясь по длине линии (true).

Пример:

```
<IndexedLineSet coordIndex="0 1 2 3 0 –1 2 6 5 1 –1"
                colorIndex="0 1 2 3 4 -1 5 6 0 1 -1"
                 colorPerVertex='true'>
  <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"></Coordinate>
  <Color color="0 0 1, 0 1 0, 0 1 1, 1 0 0, 1 0 1, 1 1 0, 1 1 1"></Color>
</IndexedLineSet>
```

Вместо задания узла Color, можно раскрасить все линии одинаково с использованием поля emissiveColor узла Material.

IndexedFaceSet

Служит для создания сложных геометрических фигур, состоящих из набора полигонов.

```
IndexedFaceSet : X3DComposedGeometryNode {
  MFNode  [in,out] attrib            []   [X3DVertexAttributeNode]
  SFNode  [in,out] color             NULL [X3DColorNode]
  SFNode  [in,out] coord             NULL [X3DCoordinateNode]
  SFNode  [in,out] fogCoord          NULL [FogCoordinate]
  SFNode  [in,out] normal            NULL [X3DNormalNode]
  SFNode  [in,out] texCoord          NULL [X3DTextureCoordinateNode]
  SFBool  []       ccw               true
  MFInt32 []       colorIndex        []   [0,∞) or -1
  SFBool  []       colorPerVertex    true
  SFBool  []       convex            true
  MFInt32 []       coordIndex        []   [0,∞) or -1
  SFFloat []       creaseAngle       0    [0,∞)
  MFInt32 []       normalIndex       []   [0,∞) or -1
  SFBool  []       normalPerVertex   true
  SFBool  []       solid             true
  MFInt32 []       texCoordIndex     []   [-1,∞)
}
```

Описание полей:

- normal - набор нормалей для полигонов (в качестве значения применяется дочерний узел Normal).
- colorPerVertex - цвета соответствуют полигонам (если значение "false") или вершинам ("true") (см. описание общих полей).
- convex - если в фигуре присутствуют невыпуклые (неплоские, самопересекающиеся) полигоны, то значением данного поля следует указать false (иначе результат работы браузера не определён) (см. описание общих полей).
- coordIndex - соответствие вершин полигонам – список индексов координат вершин с маркером окончания (-1) для каждого полигона. 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. Замыкать координаты полигонов (совмещать конечную точку с начальной) необязательно, это осуществляется автоматически.

- colorIndex – соответствие элементов массива цветов вершинам или полигонам (в зависимости от значения colorPerVertex).
- creaseAngle – если не задан узел Normal, 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 (см. описание общих полей).
- normalIndex – перечисляются номера нормалей на соответствие вершинам или поверхностям (в зависимости от значения normalPerVertex) – по такому же принципу, как в colorIndex.
- normalPerVertex - нормали соответствуют полигонам ("false") или вершинам ("true").

```
<IndexedFaceSet ccw="true" colorPerVertex="true" convex="true"
                creaseAngle="0" solid="false"
                coordIndex="0 4 7 3 -1 5 6 2 1 -1 4 5 6 7 -1 6 7 3 2 -1"
                colorIndex="0 1 2 3 -1 4 5 6 0 -1 1 2 3 4 -1 5 6 0 1 -1">
  <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"></Coordinate>
  <Color color="0 0 1, 0 1 0, 0 1 1, 1 0 0, 1 0 1, 1 1 0, 1 1 1"></Color>
</IndexedFaceSet>
```

ElevationGrid

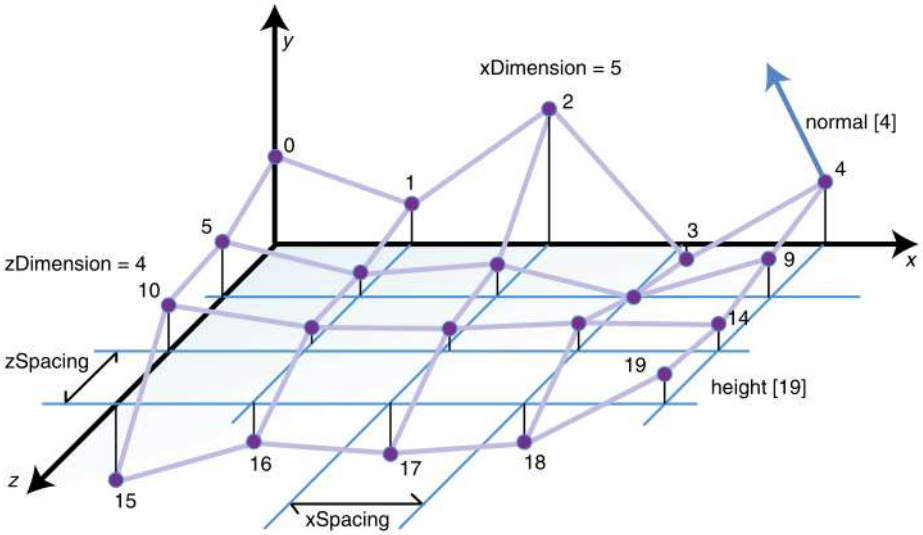
Служит для создания поверхностей из набора четырехугольников, определяя сетку возвышенностей над горизонтальной плоскостью.

```
ElevationGrid : X3DGeometryNode {
  MFNode  [in,out] attrib          []   [X3DVertexAttributeNode]
  SFNode  [in,out] color           NULL [X3DColorNode]
  SFNode  [in,out] fogCoord        NULL [FogCoordinate]
  SFNode  [in,out] normal          NULL [X3DNormalNode]
  SFNode  [in,out] texCoord        NULL [X3DTextureCoordinateNode]
  SFBool  []       ccw             true
  SFBool  []       colorPerVertex  true
  SFFloat []       creaseAngle     0    [0,∞)
  MFFloat []       height          []   (-∞,∞)
  SFBool  []       normalPerVertex true
  SFBool  []       solid           true
  SFInt32 []       xDimension      0    [0,∞)
  SFFloat []       xSpacing        1.0  (0,∞)
  SFInt32 []       zDimension      0    [0,∞)
  SFFloat []       zSpacing        1.0  (0,∞)
}
```

Описание полей:

- xDimension - | - эти переменные определяют размерность сетки возвышенностей.
- zDimension- Соответственно массив height должно содержать xDimension*zDimension | элементов
- xSpacing- | -здесь определяется расстояние между соседними вершинами в метрах
- zSpacing- | соответственно по осям X и Z

- height - определяет массив 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).




```
<ElevationGrid xDimension="7" zDimension="6"
               height="1.5, 10, 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">
</ElevationGrid>
```

Узел ElevationGrid также может содержать дочерние узлы Color (или ColorRGBA) и Normal. Поскольку индексирование цветов и нормалей не производится, при необходимости применения к полигонам или вертексам сетки возвышенностей различных цветов или векторов нормалей, массивы дочерних объектов Color (или ColorRGBA) и Normal должны содержать столько компонентов, возможно повторяющихся, сколько необходимо, причем в порядке их применения к полигонам или вертексам.

Поскольку объект, образуемый узлом ElevationGrid, состоит из четырехугольников, довольно легко можно столкнуться с проблемой неплоских полигонов, что может привести к непредсказуемости триангуляции. Данный эффект можно смягчить использованием для поверхности гладкого затенения путем повышения значения поля creaseAngle.

Extrusion

Несмотря на присутствие в спецификации X3D, узел экструзии не поддерживается в X3DOM.
https://github.com/x3dom/x3dom/issues/53

IndexedTriangleSet

Служит для создания сложных геометрических фигур, состоящих из треугольников.

Треугольники нуждаются в минимальном количестве предварительной обработки перед рендерингом и, как правило, заранее триангулированные объекты отрисовываются наиболее эффективно, что может быть критичным для больших или очень детализированных объектов.

По принципу использования узел IndexedTriangleSet напоминает IndexedFaceSet и по сути является его упрощенной версией, позволяющей определять в качестве полигонов только треугольники. Так же как и IndexedFaceSet, содержит дочерние узлы Color (или ColorRGBA) и Coordinate.

В узлах IndexedTriangleSet, IndexedTriangleStripSet значение поля colorPerVertex игнорируется и устанавливается в true. Следовательно, цвета в массивах дочерних узлов Color или ColorRGBA соответствуют вертексам.

Описание полей:

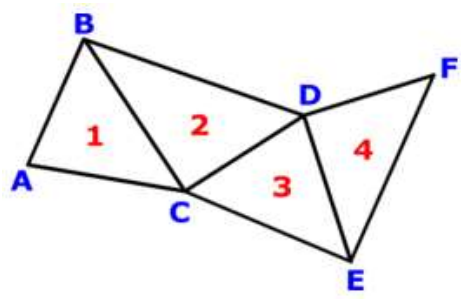
index – перечисление троек индексов координат из массива дочернего узла Coordinate. Каждая тройка индексов образует новый треугольник. Разделитель (-1) не требуется. Данный индекс также индексирует цвета из массива дочернего узла Color (или ColorRGBA).

```
<IndexedTriangleSet index='0 1 2 3 4 5 6 7 8' solid='false'>
  <Coordinate point='-4 1 3 -2 2 1.5 -3 4 0.5 -2 3 1.5 0 4 0.0 2 3 1.5 5 5 -2.5 4 3 1.5 6 4 2.0
```

```
<IndexedTriangleSet index="0 1 2 3 4 5 6 7 8" solid="false">
  <Coordinate point="-4 1 3, -2 2 1.5, -3 4 0.5, -2 3 1.5,
                     0 4 0, 2 3 1.5, 5 5 -2.5, 4 3 1.5, 6 4 2"></Coordinate>
  <ColorRGBA color="0 .8 0 1, 0 1 1 1, 1 0 0 0, 1 .5 0 0,
                   .8 0 1 0, 1 1 0 1, .6 .3 .1 0, 1 0 .5 1, 0 1 .5 0"></ColorRGBA>
</IndexedTriangleSet>
```

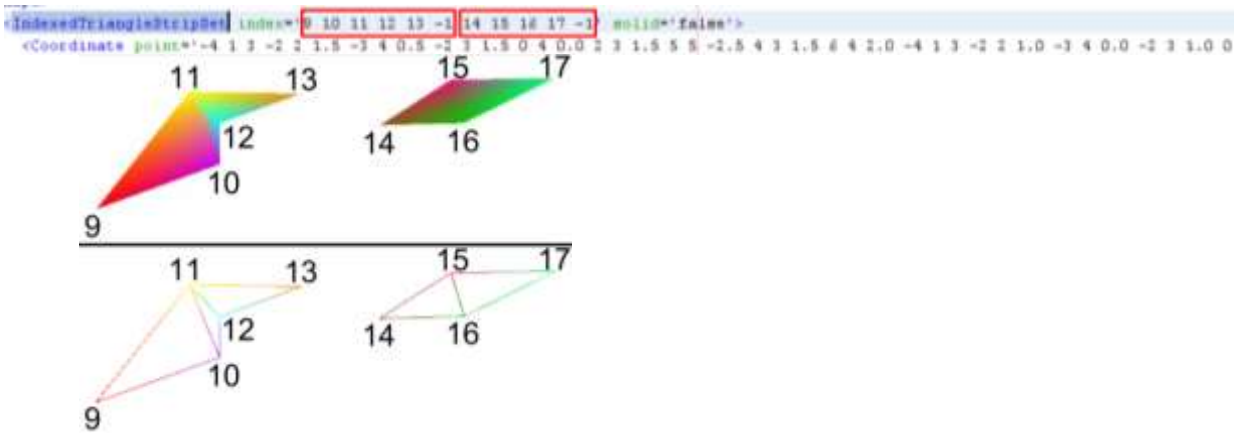
IndexedTriangleStripSet

Служит эффективным способом создания объектов, состоящих из полосы треугольников.



Описание полей:

index – перечисление индексов координат из массива дочернего узла Coordinate. Когда формирование полосы закончено, ставится разделитель (-1), после чего можно определить новую полосу. Данный индекс также индексирует цвета из массива дочернего узла Color (или ColorRGBA).



В каждой части массива index первые три элемента определяют первый треугольник, а каждый последующий элемент – последующий треугольник. Последующие треугольники образуются добавлением текущей координаты к двум предыдущим. Количество элементов массива между разделителями не может быть менее трех.

DEF и USE

С помощью атрибута DEF можно задавать уникальное имя для объекта сцены и впоследствии использовать этот объект повторно с помощью указания заданного имени в качестве значения атрибута USE узла того же типа. Это сокращает код и вычислительные затраты, т.к. объект создается единожды. Помимо этого, осмысленные DEF имена позволяют легче ориентироваться в коде сцены.

Имя, задаваемое с помощью DEF, должно быть цифробуквенным и начинаться с буквы. Допускается только латиница. Имена регистрозависимы.

```
<Scene>
  <Transform DEF="LeftCube" translation="-2 0 0">
    <Shape DEF="MyCube">
    <Appearance>
      <Material diffuseColor="1 0 0"></Material>
    </Appearance>
    <Box></Box>
  </Shape>
  </Transform>
  <Transform DEF="RightCube" translation="2 0 0">
    <Shape USE="MyCube"></Shape>
  </Transform>
</Scene>
```

Чаще всего копированию подвергаются узлы Shape, Material, Appearance, Group, Transform.

Гиперссылки

К объектам сцены или их группам можно привязать гиперссылки, которые активируются при нажатии на объект.

Для этого служит узел Anchor. Он группирует дочерние узлы и связывает с ними гиперссылку.

Описание полей:

parameter – параметры перехода. Например, значение `parameter="target='_self'"` указывает, что переход осуществляется в текущей вкладке (по умолчанию открывается новая).

url – URL, на который осуществляется переход.

В примере у Anchor два дочерних узла, переход на сайт осуществляется при нажатии на любой из них.

```
<Scene>
  <Anchor url="http://guap.ru" parameter="target='_self'">
    <Shape DEF="Cube">
      <Box>
      </Box>
      <Appearance>
        <Material></Material>
      </Appearance>
    </Shape>
    <Transform translation="3 0 0">
      <Shape USE="Cube">
    </Shape>
```

```
    </Anchor>
  </Scene>
```

Встраивание

В сцену X3DOM можно встроить внешний X3D-файл. Объекты из встроенной сцены появятся в основной, как если бы они были описаны в текущем файле. Для этого используется узел Inline.

Описание полей:

url – набор путей к файлу сцены. Путей может быть несколько, в этом случае достигается большая надежность загрузки файла, поскольку по некоторым из них он может быть недоступен.

load – загружать ли файл сцены автоматически (true/false). Атрибут может быть выставлен в true динамически по какому-то событию, тогда подгрузка произойдет в этот момент, а не при начальной загрузке сцены.

```
<Scene>
  <Shape>
    <Sphere></Sphere>
    <Appearance>
      <Material></Material>
    </Appearance>
  </Shape>
  <Transform translation="0 3 0">
    <Inline url='"cube.x3d","http://mysite.com/x3d/cude.x3d"'>
  </Transform>
</Scene>
```

Во встраиваемом файле должен присутствовать корневой элемент <Scene>, чтобы он мог быть корректно разобран как файл X3D. Для вышеприведенного примера файл cube.x3d может иметь вид:

```
<Scene>
  <Shape
    <Box></Box>
    <Appearance>
      <Material></Material>
     </Appearance>
   </Shape>
</Scene>
```

Группирование

К группирующим узлам относятся Group, Anchor, Transform

Все они способны включать в себя в качестве дочерних наборы других узлов сцены. При этом все дочерние элементы объединяются в единую группу.

Группирование позволяет достичь следующих целей:

- структурирование отдельных элементов сцены логическим образом;
- объединение связанных элементов для упрощения работы с ними;
- поддержание общей для группы системы координат, в пределах которой объекты могут быть легко позиционированы и ориентированы друг относительно друга;
- упрощение тиражирования групп связанных объектов.

Группирующие узлы могут быть вложены друг в друга:

```
<Group DEF="Gr1">
  ...
  <Transform DEF="Tr">
    ...
    <Group DEF="Gr2">
    ...
    </Group>
  </Transform>
</Group>
```

Освещение

Для проектирования реалистичных трехмерных сцен недостаточно разработать геометрические свойства объектов и свойства их внешнего вида. Важную роль играет освещение сцены набором источников, которые могут не только повысить реалистичность сцены, но и добиться интересных визуальных эффектов.

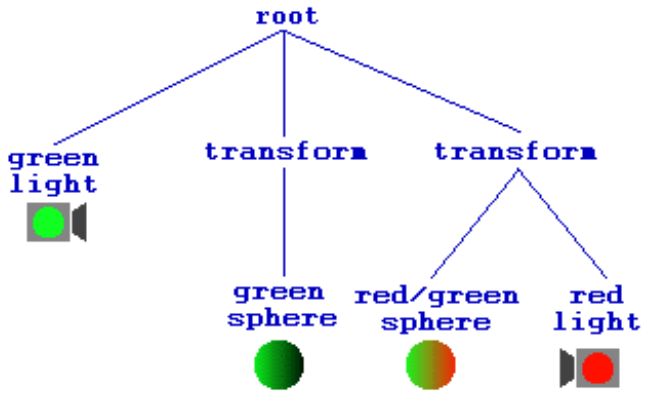
Принцип действия освещения в X3D подобен физическим явлениям реального мира, но является их сильно упрощенной аппроксимацией, поскольку рендеринг осуществляется в реальном времени, и вычислительная нагрузка при вычислении освещения не может быть слишком интенсивной. Виртуальные источники освещения испускают прямые виртуальные лучи, имеющие некоторые характеристики цвета и интенсивности, часть из которых затем отражаются от поверхностей объектов согласно вычисленным или заданным для них нормалям, изменяя свои свойства в зависимости от особенностей материала поверхности или ее текстуры, и поступает в точку наблюдения. Так формируется кадр, видимый пользователю. Среди упрощений модели освещения реального мира следует отметить отсутствие отражений и преломления.

Узлы-источники освещения не создают какой-либо сопутствующей геометрии, единственным свидетельством их наличия в сцене является освещенность других объектов. Если необходимо визуальное представление источника, необходимо разработать его вручную.

Описание общих полей:

- ambientIntensity – доля источника в рассеянном освещении сцены за счет отражения от объектов (от 0 до 1). Рассеянное освещение не имеет направленности и освещает все поверхности одинаково. В качестве примера можно привести освещенность интерьера комнаты в дневные часы (даже при отсутствии прямой видимости солнца).
- color – цвет освещения.
- intensity – интенсивность освещения (от 0 до 1).
- on – источник света включен (при значении true) или выключен (при значении false).
- global – является ли источник глобальным (true) или локальным (false). Глобальный источник освещает все объекты сцены, локальный – только содержимое группирующего узла, в котором он размещен. Данный прием может использоваться для избегания ненужного освещения (например, когда источник освещения внутри комнаты освещает предметы снаружи ее), а также для уменьшения вычислительных затрат путем сокращения количества источников, которые нужно принимать во внимание при расчете освещенности поверхности.

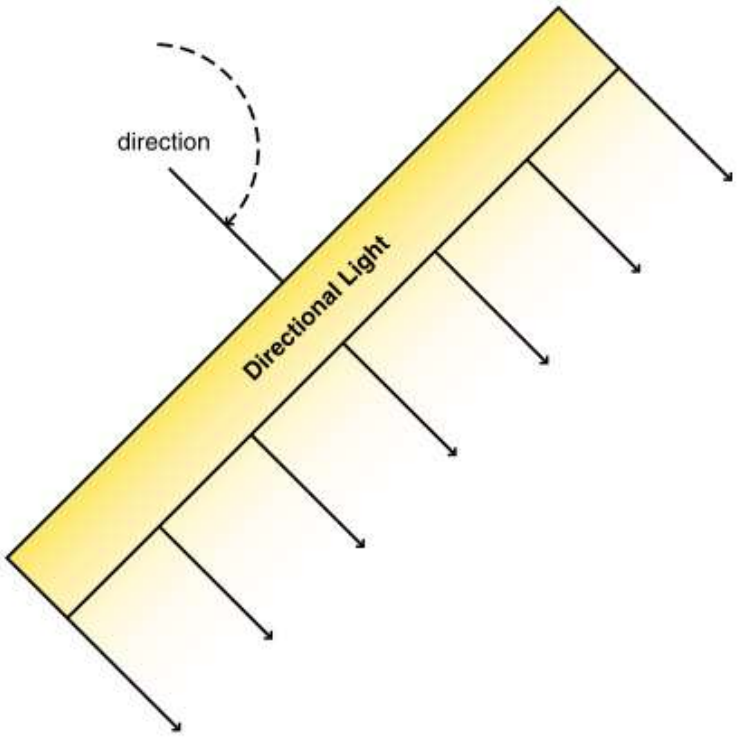
По графу сцены это можно проиллюстрировать, например, так:



DirectionalLight

Создает источник направленного освещения.
При помощи этого узла задается освещение параллельными лучами в указанном направлении. По умолчанию направление совпадает с отрицательным направлением оси z. Источник предполагается бесконечно удаленным, поэтому задается только вектор направления освещения. Затухание отсутствует.

Описание полей:
- direction – вектор направления освещения.

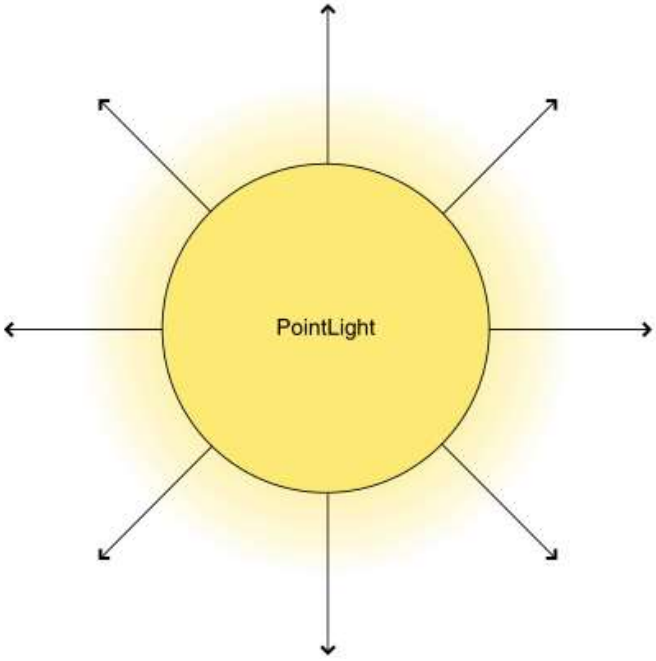


Headlight

Headlight не является узлом X3D, а представляет собой встроенный в браузер источник освещения типа DirectionalLight, который фиксирован в положении и ориентации текущей точки наблюдения пользователя. Источник включен по умолчанию и может быть отключен булевым полем headlight узла NavigationInfo.

PointLight

Задает точечный источник света, который излучает во всех направлениях. Интенсивность освещения, получаемого от PointLight, зависит от расстояния источника до объекта, поскольку используется затухание.



Описание полей:

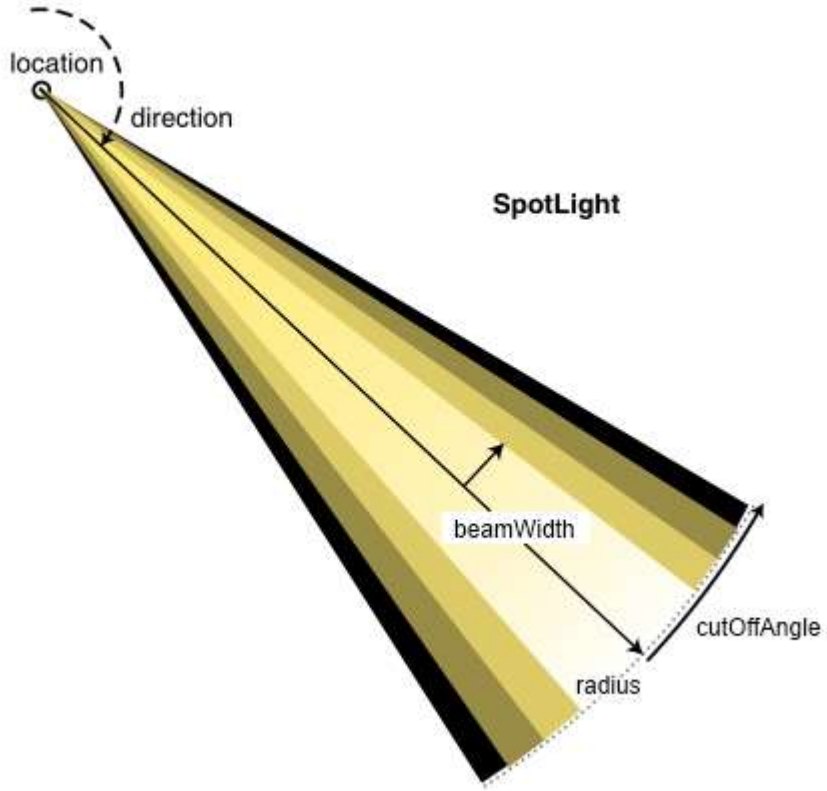
- attenuation – затухание. Определяет, как быстро будет падать интенсивность освещения при удалении от местоположения источника. Три числа, указываемые в качестве значения этого поля (коэффициенты константного, линейного и квадратичного затухания), используются для вычисления интенсивности I_r относительно исходной I_0 на расстоянии r от центра по формуле:

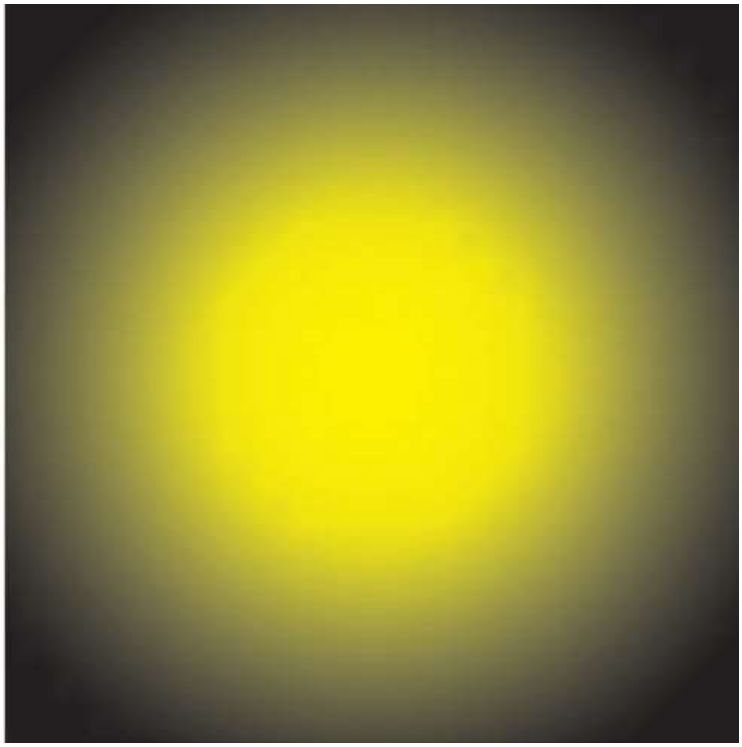
$$I_r = \frac{I_0}{\max(1, attenuation[0] + attenuation[1] \times r + attenuation[2] \times r^2)}$$

- location – координаты местоположения источника.
- radius – максимальное эффективное расстояние освещения.

SpotLight

Определяет источник освещения, который имеет свое местоположение и светит в определенном направлении коническим пучком лучей. Результатом освещения является световое пятно с размытием. Есть возможность гибко задавать параметры данного светового пятна. Полная интенсивность достигается в пределах конуса, определяемого углом beamWidth, затем она линейно убывает до нуля в пределах конуса с углом cutOffAngle. Такой подход позволяет достичь эффекта мягкого освещения.





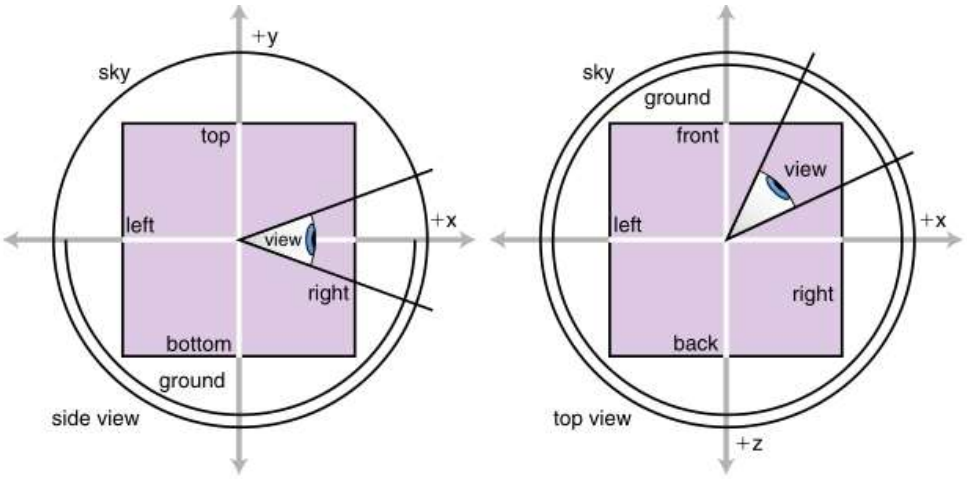
Описание полей:
beamWidth – угол раствора внутреннего конуса, в пределах которого интенсивность равна intensity.
cutOffAngle – угол раствора внешнего конуса. Снаружи него интенсивность равна 0, а между двумя конусами она линейно интерполируется.
По умолчанию beamWidth > cutoffAngle, что дает пятно с неразмытыми краями.

Background

Позволяет задавать для сцены фон, состоящий из набора цветов или панорамных текстур.

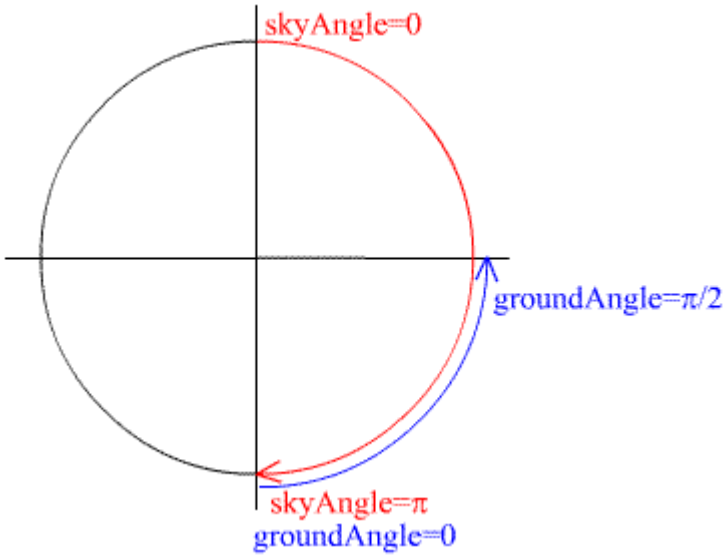
Фон в сценах X3D может быть реализован либо в форме набора изображений формирующих панораму, либо в виде набора цветов, представляющих небо и землю. Возможно совмещение обоих подходов.

Фон, задаваемый набором цветов, располагается на сфере бесконечного радиуса. Текстурированный фон располагается на внутренней поверхности куба, имеющего бесконечные размеры. При использовании текстурированного фона он перекрывает фон, состоящий из набора цветов. Также цвета земли перекрывают цвета неба, заданные для того же угла сферы.



Принцип задания цветов для фона таков. Для неба задается ряд углов от 0 до π, где 0 соответствует верхней точке сферы, а π – нижней, которые ограничивают горизонтальные кольцеобразные участки небесной сферы. Для каждого участка сферы задается цвет. Для земли подход практически идентичен, но углы варьируются от 0 до $\frac{\pi}{2}$, где 0 соответствует нижней точке сферы, $\frac{\pi}{2}$ – горизонту. И для земли, и для неба, цвета соседних участков линейно интерполируются, создавая эффект реалистичности.

В случае если цвета земли не заданы, цвета неба используются для всего фона.



Описание полей:

- skyAngle – массив, содержащий неубывающее перечисление углов внутри сферы (их количество на 1 меньше, чем количество цветов в skyColor). Диапазон изменения углов от 0 до π, 0 соответствует верхней точке сферы, π – нижней.
- skyColor – последовательно определяются цвета, в которые будут покрашены участки небесной сферы, ограниченные углами, перечисленными в массиве skyAngle. Первый цвет соответствует верхней точке сферы. Цвета между кольцами линейно интерполируются.
- groundAngle – то же, что и skyAngle, диапазон изменения углов от 0 до $\frac{\pi}{2}$, 0 соответствует нижней точке сферы, $\frac{\pi}{2}$ – горизонту.
- groundColor – то же, что и skyColor, но для groundAngle.
- backUrl – пути к файлу с изображением для задней стенки куба.
- bottomUrl – пути к файлу с изображением для нижней стенки куба.
- frontUrl – пути к файлу с изображением для передней стенки куба.
- leftUrl – пути к файлу с изображением для левой стенки куба.
- rightUrl – пути к файлу с изображением для правой стенки куба.
- topUrl – пути к файлу с изображением для верхней стенки куба.

Fog

Позволяет добавить в сцену эффект тумана или ночной мглы (при применении тумана черного цвета). Цвет тумана постепенно замещает цвет объектов при удалении их от наблюдателя. Поэтому для реалистичности важно, чтобы цвет тумана совпадал с цветом фона.

Описание полей:

- color – цвет тумана.
- fogType – тип тумана - “LINEAR” (линейный) или “EXPONENTIAL” (экспоненциальный).
- visibilityRange – расстояние от наблюдателя, при превышении которого объекты полностью скрыты туманом (их можно не отрисовывать).

Навигация

Продуманная навигация пользователя по трехмерной сцене является важным компонентом интерактивности. Стандарт X3D предоставляет расширенные возможности управления навигацией: задание набора точек наблюдения, контроль параметров перемещения пользователя по сцене.

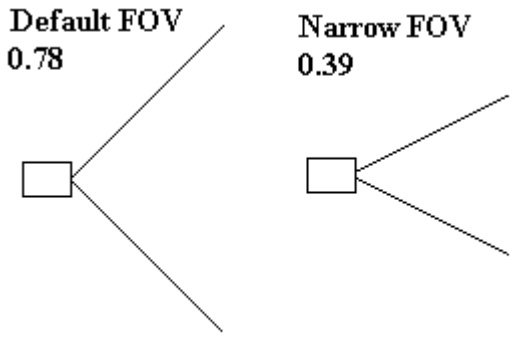
Viewpoint

Позволяет задавать местоположение и ориентацию точек наблюдения. Точку наблюдения можно отождествить с камерой, через которую пользователь смотрит на мир.

Точек наблюдения в сцене может быть произвольное количество, каждая из них может обладать своими свойствами и располагаться удобно с точки зрения целей, которые хочет достичь разработчик сцены. Некоторые из точек наблюдения могут быть анимированы. Переключение между точками осуществляется с помощью клавиш PgUp/PgDn.

Описание полей:

- fieldOfView – угол обзора камеры в радианах. Анимация этого поля может давать интересные эффекты.



- orientation – начальная ориентация точки наблюдения.
- position – начальная позиция точки наблюдения.
- description – описание точки наблюдения (выводится в браузере).
- centerOfRotation – координата точки в пространстве, вокруг которой происходит вращение камеры, если в узле NavigationInfo установлен режим EXAMINE. Если пользователь выбирает режим LOOKAT и выбирает другую точку значение этого поля изменяется.

NavigationInfo

Позволяет задать параметры перемещения пользователя по сцене.

Описание полей:

- avatarSize – определяет размеры аватара (представления пользователя) пользователя. Представляет собой тройку вещественных чисел.

a – размер по горизонтали – используется для проверки на столкновение с другими объектами.
b – размер по вертикали – определяет, насколько высоко над объектами находится позиция наблюдения.
c – максимальная высота объектов, которые можно “перешагнуть”. По этому признаку можно отличить к примеру, лестницу, на которую можно подняться, от стены, которая является непреодолимым препятствием (в режиме “WALK”).

- headlight – включена ли подсветка для сцены по умолчанию в виде направленного источника освещения белого цвета единичной интенсивности, сонаправленного с направлением взгляда пользователя. При значении false освещение по умолчанию выключено, необходимо использовать создаваемые вручную источники освещения.
- speed – скорость перемещения по сцене (м/с).
- type – тип навигации, который будет установлен в браузере. Можно указывать несколько типов навигации, среди которых пользователь может выбрать наиболее удобный для него.
  Возможные значения параметра:
  `"ANY"` – доступны все режимы навигации по выбору пользователя.
  `"WALK"` – ходьба. Пользователь перемещается по геометрии земной поверхности, если она задана, проверка на столкновения не дает ему провалиться сквозь землю. Гравитация включена.
  `"EXAMINE"` – изучение. Движением мыши вращается вся сцена вокруг центра вращения текущей точки наблюдения.
  `"FLY"` – полет. Отличается от ходьбы отсутствием гравитации.
  `"LOOKAT"` – рассматривание. Пользователь может указывать произвольные объекты для рассматривания, что влечет перемещение и переориентацию камеры и смену центра ее вращения.
  `"NONE"` – навигация отключена, возможно только переключение между точками наблюдения.
- visibilityLimit – определяет, как далеко пользователь может видеть. Рендеринг за пределами этого значения браузер не проводит. Значение по умолчанию соответствует бесконечному пределу.
- transitionType – dfdf

Collision

Позволяет задать, проходит ли аватар (представление пользователя) сквозь геометрические объекты сцены. Является группирующим узлом и определяет соответствующие свойства для всех своих дочерних узлов.

Описание полей:

- enabled – при значении true проверка на столкновения аватара с объектами осуществляется, при значении false нет.
- bboxCenter – центр bounding box.
- bboxSize – размеры bounding box.
- proxy – объект, указанный здесь, не отображается, но осуществляется проверка на столкновение аватара с ним. Может быть полезно для снижения вычислительной нагрузке при просчете столкновений с объектами сложной формы. Так, можно определить вокруг объекта невидимый параллелепипед и обнаруживать столкновения с ним, а не с самим объектом. Для этого следует указать в дочернем объекте атрибут `containerField="proxy"`.

```
<Collision DEF="MyCollisionNode" enabled="true">
  <Group DEF="MyCollidableGroup"> … </Group>
  <Shape DEF="MyHiddenProxy" containerField="proxy"> … </Shape>
</Collision>
```

Анимация

События

Узлы X3D помимо полей, задаваемых в design-time (при проектировании сцены) определяют события (входные и выходные), которые могут быть связаны между собой и вызывать друг друга с целью получения анимации. Каждое событие имеет тип, связанный с ним (SFFloat, MFString,…). Соединены могут быть только события одного типа.

В спецификации для каждого поля указывается тип доступа к нему.

Существует 4 типа доступа:

[] – поле не может участвовать в генерации или получении событий. Значение таких полей задается в design-time.

[in] – поле не задается в design-time, а является входным событием, определяющим поведение узла. Может участвовать в маршрутах как атрибут toField.

[out] – поле не задается в design-time, а является выходным событием, определяющим поведение узла. Может участвовать в маршрутах как атрибут fromField.

[in,out] – поле может задаваться в design-time и имеет два события, связанные с ним: set_имя_поля (входное) и имя_поля_changed (выходное). При этом если задействуется извне событие set_имя_поля, то узел генерирует соответствующее ему событие имя_поля_changed.

Для того чтобы одни события вызывали другие, необходимо связывать их между собой маршрутами. Маршрут задается с помощью узла ROUTE.

В атрибутах узла ROUTE указывается:

- fromNode – DEF-имя узла, являющегося источником выходного события;
- fromField – выходное событие этого узла;
- toNode – DEF-имя узла, являющегося приемником входного события;
- toField – выходное событие этого узла.

Пример:

```
<ROUTE fromNode="TimeClock" fromField="fraction_changed"
       toNode="PosInt" toField="set_fraction">
</ROUTE>
```

TimeSensor

Таймер. Генерирует события в заданном интервале времени.

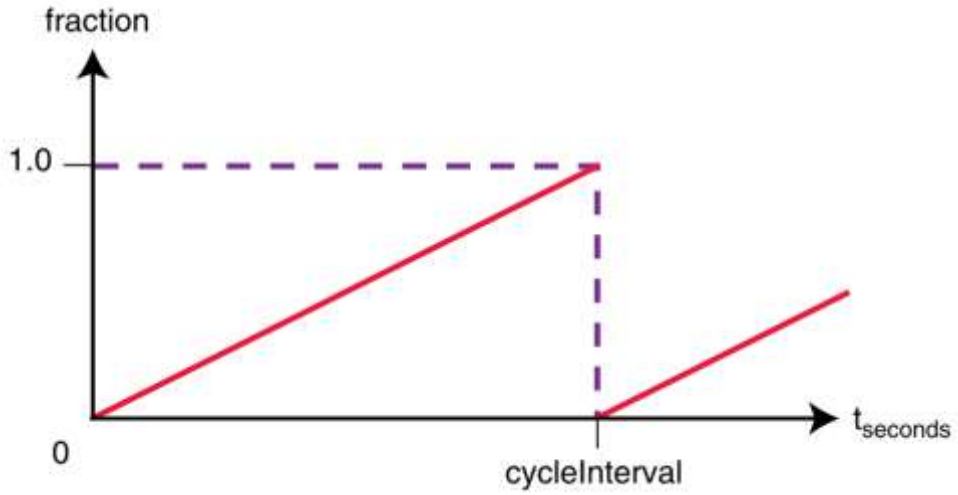
```
TimeSensor : X3DTimeDependentNode, X3DSensorNode {
  SFTime  [in,out] cycleInterval    1     (0,∞)
  SFBool  [in,out] enabled          true
  SFBool  [in,out] loop             false
  SFTime  [in,out] pauseTime        0     (-∞,∞)
  SFTime  [in,out] resumeTime       0
  SFTime  [in,out] startTime        0     (-∞,∞)
  SFTime  [in,out] stopTime         0     (-∞,∞)
  SFTime  [out]    cycleTime
  SFTime  [out]    elapsedTime
  SFFloat [out]    fraction_changed
```

```
  SFBool  [out]    isActive
  SFBool  [out]    isPaused
  SFTime  [out]    time
}
```

Описание полей:

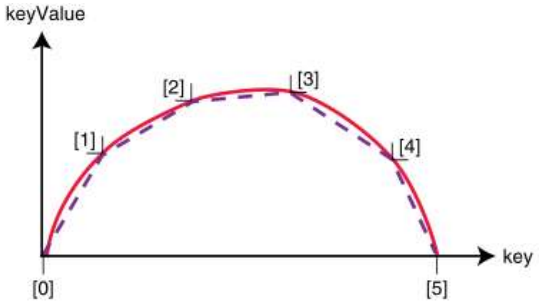
- cycleInterval – интервал цикла (секунды).
- loop – зациклен таймер или нет (true/false). При значении true события будут генерироваться циклически, в противном случае они будут генерироваться только в течение первого интервала цикла.
- fraction_changed – выходное событие, представляющее собой непрерывный поток сигналов, необходимый для интерполяторов. Событие генерируется постоянно, и так быстро, как возможно. Значение этого события интерполируется во время интервала цикла от 0 до 1 (при достижении таймером значения, кратного интервалу цикла, значение fraction_changed = 1).
- enabled – таймер включен (true) или отключен (false)
- startTime – время включения таймера.
- stopTime – время выключения таймера.
- cycleTime – это событие генерируется каждый раз, когда таймер достигает интервала цикла, и имеет значение текущего времени.
- time – генерируется так же, как и fraction_changed, и имеет значение текущего времени.
- isActive – генерируется, когда таймер начинает работать или останавливается (true – таймер запущен, false – прекратил работу).
- pauseTime – время, когда таймер будет поставлен на паузу.
- resumeTime – время, когда таймер возобновит работу.

Выходное событие fraction_changed представляет собой пилообразную функцию.



Узлы-интерполяторы

Модель X3D предполагает, что любая анимация во времени может быть аппроксимирована кусочно-линейной функцией. Это позволяет разработчику вручную задавать настолько детализированную функцию анимации, насколько это необходимо.



На рисунке кривую аппроксимируют 6 точек, соединенных между собой отрезками. Иными словами, между ключевыми точками осуществляется линейная интерполяция.

Для реализации этой концепции используются узлы-интерполяторы, осуществяющие линейную интерполяцию анимируемой величины согласно функции, заданной таблицей значений. Таблица значений задается разработчиком в design-time.

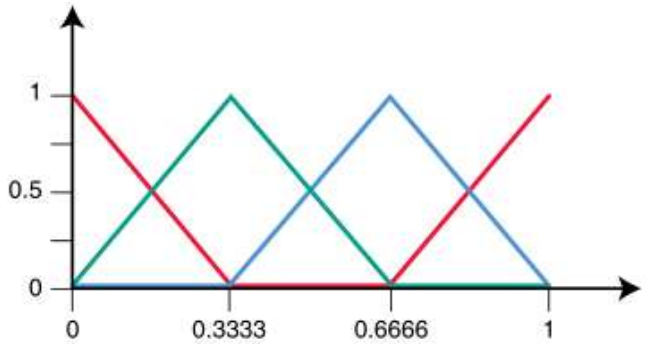
Вышеприведенному рисунку соответствует таблица

key	0	0.2	0.4	0.6	0.8	1
keyValue	0	5	8	9	4	0

Это реализуется интерполятором скалярной величины следующего вида:

```
<ScalarInterpolator key="0 0.2 0.4 0.6 0.8 1"
                    keyValue="0 5 8 9 4 0">
</ScalarInterpolator>
```

В качестве интерполируемого значения не обязательно должна выступать скалярная величина. Им может быть, к примеру, трехкомпонентное значение цвета:



Рисунку соответствует код

```
<ColorInterpolator key="0, 0.3333, 0.6666, 1"
                   keyValue="1 0 0, 0 1 0, 0 0 1, 1 0 0">
</ColorInterpolator>
```

Существует множество узлов-интерполяторов, различающихся типом данных интерполируемой величины (тип данных поля keyValue и выходного события value_changed).

Описание общих полей:

key и keyValue – задают функцию линейной интерполяции. Массив key перечисляет в неубывающем порядке границы относительных временных интервалов (в диапазоне [0..1]), которым соответствуют значения интерполируемой величины в массиве keyValue.

В момент времени, равный элементу массива key, выходное событие value_changed принимает значение, равное соответствующему элементу массива keyValue. В пределах интервалов, определяемых границами массива key, для значений value_changed осуществляется линейная интерполяция.

set_fraction – входное событие, принимаемое с таймера, значение которого в течение интервала таймера пробегает от 0 до 1 и сравнивается со значениями из key, и при совпадении осуществляется переход к следующему участку интерполяции. Соответственно, длительность полного цикла интерполяции равна интервалу цикла таймера.

value_changed – выходное событие, значение которого интерполируется между значениями из keyValue.

ColorInterpolator (Интерполятор цвета)

Осуществляет интерполяцию между значениями цвета (тройками вещественных чисел 0..1).

```
<Scene>
  <TimeSensor DEF="Time" cycleInterval="5" loop="true"> </TimeSensor>
  <ColorInterpolator DEF="ColInt" key="0 0.33 0.66 1"
                     keyValue="1 0 0, 0 1 0, 0 0 1, 1 0 0">
  </ColorInterpolator>
  <Shape>
    <Appearance>
      <Material DEF="Mat" diffuseColor="1 0 0"></Material>
    </Appearance>
    <Cylinder></Cylinder>
  </Shape>
  <ROUTE fromNode="Time" fromField="fraction_changed"
         toNode="ColInt" toField="set_fraction"></ROUTE>
  <ROUTE fromNode="ColInt" fromField="value_changed"
         toNode="Mat" toField="set_diffuseColor"></ROUTE>
</Scene>
```

ScalarInterpolator

Осуществляет интерполяцию между значениями скалярной величины (одиночного вещественного числа, тип SFFloat).

```
<Scene>
  <TimeSensor DEF="Time" cycleInterval="5" loop="true"> </TimeSensor>
  <ScalarInterpolator DEF="ScInt" key="0 .5 1"
                      keyValue=".1 .9 .1"></ScalarInterpolator>
  <Shape>
    <Appearance>
      <Material DEF="Mat" transparency="0"></Material>
    </Appearance>
    <Box></Box>
  </Shape>
  <ROUTE fromNode="Time" fromField="fraction_changed"
         toNode="ScInt" toField="set_fraction"></ROUTE>
  <ROUTE fromNode="ScInt" fromField="value_changed"
         toNode="Mat" toField="set_transparency"></ROUTE>
</Scene>
```

CoordinateInterpolator

Осуществляет интерполяцию между значениями набора координат в трехмерном пространстве (тип MFVec3f).

```
<Scene>
  <TimeSensor DEF="Time" cycleInterval="5" loop="true"></TimeSensor>
  <CoordinateInterpolator DEF="CoordInt" 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">
  </CoordinateInterpolator>
  <Shape>
    <Appearance>
      <Material></Material>
    </Appearance>
    <IndexedFaceSet coordIndex="2 3 0 -1 2 0 1 -1 3 2 1 -1 3 1 0 -1">
      <Coordinate DEF="Coord" point="0 1 1, 0 1 -1, -1 -1 0, 1 -1 0"></Coordinate>
    </IndexedFaceSet>
  </Shape>
  <ROUTE fromNode="Time" fromField="fraction_changed"
         toNode="CoordInt" toField="set_fraction"></ROUTE>
  <ROUTE fromNode="CoordInt" fromField="value_changed"
         toNode="Coord" toField="set_point"></ROUTE>
</Scene>
```

OrientationInterpolator

Осуществляет интерполяцию между значениями вектора ориентации в пространстве (тип SFRotation).

```
<Scene>
  <TimeSensor DEF="Time" cycleInterval="5" loop="true"></TimeSensor>
  <OrientationInterpolator DEF="OrientInt" key="0 0.5 1"
                           keyValue="0 0 1 0, 0 0 1 -3.14, 0 0 1 -6.28">
  </OrientationInterpolator>
  <Shape>
    <Appearance>
      <Material></Material>
    </Appearance>
    <Box></Box>
  </Shape>
  <Transform DEF="Trans">
    <Transform DEF="Trans2" translation="0 -3 0">
      <Shape>
        <Appearance>
          <Material></Material>
        </Appearance>
        <Cone></Cone>
      </Shape>
    </Transform>
  </Transform>
  <ROUTE fromNode="Time" fromField="fraction_changed"
```

```
        toNode="OrientInt" toField="set_fraction"></ROUTE>
  <ROUTE fromNode="OrientInt" fromField="value_changed"
        toNode="Trans" toField="set_rotation"></ROUTE>
</Scene>
```

PositionInterpolator

Осуществляет интерполяцию между значениями одиночной координаты (тип SFVec3f).

```
<Scene>
  <TimeSensor DEF="Time" cycleInterval="5" loop="true" enabled="true"></TimeSensor>
  <PositionInterpolator DEF="PosInt" key="0 0.23 0.5 0.75 1"
                        keyValue="-3 -3 0, -3 3 0, 3 3 0, 3 -3 0, -3 -3 0">
  </PositionInterpolator>
  <Shape>
    <Appearance>
      <Material></Material>
    </Appearance>
    <Box></Box>
  </Shape>
  <Transform DEF="Trans" translation="-3 -3 0">
    <Shape>
      <Appearance>
        <Material></Material>
      </Appearance>
      <Sphere radius="0.25"></Sphere>
    </Shape>
  </Transform>
  <ROUTE fromNode="Time" fromField="fraction_changed"
        toNode="PosInt" toField="set_fraction"></ROUTE>
  <ROUTE fromNode="PosInt" fromField="value_changed"
        toNode="Trans' toField="set_translation"></ROUTE>
</Scene>
```

Порядок действий при создании анимации

1) Определите узел, который должен быть анимирован.
2) Укажите для него DEF-имя.
3) Определите поле, которое должно изменять значение и определите тип анимации.
4) Выберите интерполятор, генерирующий значения value_changed соответствующего типа. Например, PositionInterpolator генерирует события типа SFVec3f.
5) Добавьте TimeSensor, установите длительность периода анимации.
6) Создайте маршрут от поля fraction_changed таймера к полю set_fraction интерполятора.
7) Создайте маршрут от поля value_changed интерполятора к анимируемому полю нужного узла.

Пример – вращение Луны вокруг Земли

```
<Scene>
  <Background backUrl="realsky_BK.jpg" bottomUrl="realsky_DN.jpg"
              frontUrl="realsky_FR.jpg" leftUrl="realsky_LF.jpg"
              rightUrl="realsky_RT.jpg" topUrl="realsky_UP.jpg">
  </Background>
  <Transform DEF="earth" translation="0 0 0" scale="1.25 1.25 1.25">
    <Shape>
      <Appearance>
        <Material diffuseColor="1 1 1" specularColor=".2 .2 .2"></Material>
        <ImageTexture url="earth.jpg"></ImageTexture>
      </Appearance>
      <Sphere></Sphere>
    </Shape>
  </Transform>
  <Transform DEF="moon" translation="11 0 0" center="-11 0 0" scale="0.35 0.35 0.35">
    <Shape>
      <Appearance>
        <Material></Material>
        <ImageTexture url="moon.jpg"></ImageTexture>
      </Appearance>
      <Sphere></Sphere>
    </Shape>
  </Transform>
  <DirectionalLight direction="0 -1 0" intensity="0.3"></DirectionalLight>
  <TimeSensor DEF="TS_e" cycleInterval="1" loop="true"></TimeSensor>
  <OrientationInterpolator DEF="OI_e" key="0, 0.5, 1"
                           keyValue="0 1 0 0, 0 1 0 3.14, 0 1 0 6.28">
  </OrientationInterpolator>
  <ROUTE fromNode="TS_e" fromField="fraction_changed"
         toNode="OI_e" toField="set_fraction">
  </ROUTE>
  <ROUTE fromNode="OI_e" fromField="value_changed"
         toNode="earth" toField="set_rotation">
  </ROUTE>
  <TimeSensor DEF="TS_m" cycleInterval="27" loop="true"></TimeSensor>
  <OrientationInterpolator DEF="OI_m" key="0, 0.5, 1"
                           keyValue="0 1 0 0, 0 1 0 3.14, 0 1 0 6.28">
  </OrientationInterpolator>
  <ROUTE fromNode="TS_m" fromField="fraction_changed"
         toNode="OI_m" toField="set_fraction">
  </ROUTE>
  <ROUTE fromNode="OI_m" fromField="value_changed"
         toNode="moon" toField="set_rotation">
  </ROUTE>
</Scene>
```

Скрипты

Программы, написанные на языке JavaScript и выполняемые на стороне клиента, позволяют встраивать в веб-страницы произвольную логику. В частности, X3DOM открывает доступ к объектам сцены как к элементам DOM, позволяя создавать, удалять элементы, изменять значения атрибутов и т.д. Действия срабатывают как реакция на события, которые могут исходить от пользователя.

В этой части будут разобраны некоторые способы добавления интерактивности в сцену с использованием JavaScript. Полную документацию языка можно найти, например, на сайте http://learn.javascript.ru/

Работа с DOM

В языке JavaScript DOM-дерево доступно для изменения посредством объекта document.

Поиск элемента

Для того чтобы взаимодействовать с объектами сцены, у объекта document определено несколько функций поиска. Рассмотрим здесь функцию доступа к конкретному элементу document.getElementById, возвращающую элемент с некоторым id.

id – специальный атрибут, который может быть задан для любого элемента документа (в том числе для объектов X3DOM-сцены). Например,

```
<Transform id="Aeroplane" translation="0 100 0">
```

На атрибут id накладывается требование уникальности. Два узла одного документа не могут иметь одинаковые значения id.

Если в коде JavaScript вызвать функцию подобным образом:

```
var obj = document.getElementById("Aeroplane");
```

то в переменную obj будет присвоен соответствующий узел Transform.

В некотором роде id дублирует функциональность атрибута DEF. Это является следствием попытки в X3DOM срастить 2 разнородные технологии: X3D и HTML. DEF – атрибут, специфичный для X3D, который может быть задан только для объектов сцены и служит для их идентификации в целях тиражирования или создания маршрутов. id – атрибут, служащий для идентификации DOM-узлов в документе HTML с целью применения стилей и обработки скриптами.

Вполне корректно задавать для одного и того же элемента сцены атрибутам id и DEF одинаковые значения:

```
<Transform id="Aeroplane" DEF="Aeroplane">
```

Доступ к дочерним элементам

У каждого элемента DOM есть свойство, представляющее собой массив дочерних (вложенных) элементов children. Элементы индексируются с нуля.

```
<Transform id="Aeroplane" translation="0 100 0">
     <Transform id="wing1"> ... </Transform>
     <Transform id="wing2"> ... </Transform>
     <Transform id="pilot">
              <Shape id="pilot_head"> ... </Shape>
              <Shape id="pilot_legs"> ... </Shape>
     </Transform>
</Transform>
```

Вызов

```
var obj = document.getElementById("Aeroplane").children[2].children[1];
```

будет эквивалентен

```
var obj = document.getElementById("pilot_legs");
```

Доступ к родительскому элементу осуществляется посредством свойства parentNode. Так, `obj.parentNode` вернет элемент `<Transform id="pilot">`

Создание элемента

Для создания элементов служит функция document.createElement. Она создает элемент с именем, переданным ей в качестве параметра. Необходимо обратить внимание, что после создания элемент не добавляется в документ. Для этого необходимо указать позицию в DOM-дереве, которую он должен занять. Также важно задать необходимые значения атрибутов (и, возможно, дочерние узлы), без которых элемент не будет корректно функционировать.

Изменение атрибута

Чрезвычайно важную роль играет функция элемента setAttribute, которая позволяет менять значение атрибута. Поскольку все характеристики элементов X3DOM задаются атрибутами, скриптовая динамика сцены строится на модификации их значений. Первым параметром в функцию передается строка-имя атрибута, вторым – значение, которое необходимо записать в атрибут.

Чтение атрибута

Для чтения значений атрибутов используется функция getAttribute.
Пример: увеличение интервала таймера в 2 раза.

```
var oldInterval = parseFloat(document.getElementById("TS_e").getAttribute("cycleInterval"));
document.getElementById("TS_e").setAttribute("cycleInterval", oldInterval*2);
```

Функция parseFloat осуществляет преобразование строкового типа к вещественному.

Добавление/удаление потомков

Функции appendChild и removeChild вызываются от имени родительского узла и принимают в качестве параметра узел, который должен быть добавлен/удален. Узел добавляется в конец списка вложенных узлов.

Пример: создается параллелепипед и помещается в корень сцены.

```
<X3D>
    <Scene id="root">
    </Scene>
</X3D>
<script>
    var s = document.createElement("Shape");
    var b = document.createElement("Box");
    b.setAttribute("size", "1 2 2");
    b.setAttribute("id", "MyBox");
    s.appendChild(b);
    var a = document.createElement("Appearance");
    b.appendChild(a);
    var m = document.createElement("Material");
    a.appendChild(m);
    document.getElementById("root").appendChild(t);
</script>
```

Для удаления параллелепипеда можно выполнить

```
var rootNode = document.getElementById("root");
rootNode.removeChild(rootNode.children[0]);
```

Иногда случается, что родительскому элементу не назначен id, и требуется удалить элемент, зная лишь его собственный id. Для этого можно воспользоваться таким приемом:

```
var delNode = document.getElementById("MyBox");
delNode.parentNode.removeChild(delNode);
```

События

В браузере возникают различного рода события, в том числе инициированные пользователем. Для них можно назначать обработчики в виде кода на JavaScript. Рассмотрим, как можно обработать событие наведения курсора (mouseover) на элемент, уведения курсора с него (mouseout) и щелчка левой кнопкой (click) по нему.

Поскольку объекты сцены X3DOM являются полноценными DOM-элементами, они могут генерировать DOM-события, что позволяет задавать произвольные действия при, например, щелчке по объекту сцены.

Чтобы задать код, обрабатывающий событие щелчка по сфере, необходимо указать его в специальном атрибуте onclick.

```
<Shape onclick='alert("Вы щелкнули по сфере.");'>
  <Sphere></Sphere>
</Shape>
```

Здесь в обработчике DOM-события click происходит вызов функции alert, которая выводит на экран сообщение с заданным в качестве параметра текстом.

Необходимо обратить внимание, что вложенными друг в друга могут быть только кавычки различных видов (двойные/одинарные).

Аналогично задаются обработчики других событий.

Пример: при наведении курсора на сферу она окрашивается в красный, при уведении – в синий.

```
<Shape
onmouseover='document.getElementById("Mat").setAttribute("diffuseColor","1 0 0")'
onmouseout='document.getElementById("Mat").setAttribute("diffuseColor","0 0 1")'>
     <Sphere></Sphere>
     <Appearance>
              <Material id="Mat" diffuseColor="0 0 1"></Material>
     </Appearance>
</Shape>
```

Внутри обработчика событий мыши click и mouseover доступна специальная переменная-массив `event.hitPnt`, которая представляет собой трехмерный вектор-координату, в которой произошло событие (щелчок/наведение курсора). Ее можно использовать в коде, к примеру, для задания входного события для узла PositionChaser.

Пример: пользователю сообщаются координаты щелчка.

```
<Sphere onclick="alert('Вы щелкнули по сфере в позиции\n' + event.hitPnt);">
```

Функции

Код удобно организовывать в функции и привязывать к событиям по их имени. При этом повышается удобочитаемость, поскольку не требуется размещать весь код в одном атрибуте. Более того, в функции можно передавать параметры, влияющие на их выполнение.

Можно расширить пример с Землей и Луной, добавив на страницу кнопки запуска и остановки анимации небесных тел.

```
<script>
    function toggle(what)
    {
        var button=(what=="earth")
            ?document.getElementById("but_e")
            :document.getElementById("but_m");
        var TS = (what=="earth")
            ?document.getElementById("TS_e")
            :document.getElementById("TS_m");
        var TS_enabled = TS.getAttribute("enabled");
        if (TS_enabled == "true") {
            TS.setAttribute("enabled", "false");
            button.value = (what=="earth")
            ?"Анимировать Землю"
            :"Анимировать Луну";
        }
        else {
            TS.setAttribute("enabled", "true");
            button.value = (what=="earth")
                ?"Остановить Землю"
                :"Остановить Луну";
        }
    }
</script>
<input type="button" id="but_e" value="Остановить Землю"
       onclick='toggle("earth");'>
```

```
<input type="button" id="but_m" value="Остановить Луну"
       onclick='toggle("moon");'>
```

Для таймеров необходимо добавить атрибут id, чтобы скрипт мог найти их в DOM-дереве.

```
<TimeSensor id="TS_e" DEF="TS_e" cycleInterval="1" loop="true"></TimeSensor>
<TimeSensor id="TS_m" DEF="TS_m" cycleInterval="27" loop="true"></TimeSensor>
```

Последователи (Followers)

У создания анимации с помощью интерполяторов есть ряд недостатков:

1) Невозможно задать реакцию интерполятора на события времени выполнения. Опорные точки для интерполяции задаются в атрибутах узла-интерполятора во время описания сцены, исключая возможность, например, их динамического задания посредством пользовательского ввода. Таким образом, интерполятор жестко ограничен формой выходного сигнала, который задается заранее и не может быть динамически модифицирован, а анимация объекта – количеством привязанных к нему интерполяторов. Практический пример: плавное перемещение объекта в точку, указанную пользователем (event.hitPnt при щелчке мышью или наведении курсора) практически неосуществимо с использованием интерполяторов.

2) Сложность программирования реакции объектов сцены на непредсказуемый пользовательский ввод. Разработчик может предусмотреть ограниченный набор анимаций, регулируемых таймерами, запускаемыми, к примеру, по щелчку. При этом если пользователь осуществил щелчок в процессе выполнения анимации (после ее старта, но до завершения), то система может некорректно обработать такое событие, к примеру, переключив таймер и интерполятор, что может вызвать резкий рывок в анимации. Альтернативой может служить отключение реакции на щелчок на период анимации, но такое решение не всегда приемлемо с точки зрения удобства пользователя.

3) Линейность интерполяции средствами интерполяторов X3D. Очень часто в реальном мире физические явления протекают по нелинейным законам. Простейший пример – анимация перемещения объекта с учетом инерции, когда в начале пути объект должен плавно ускоряться, а в конце – замедляться. В случае, когда интерполяция линейна, позиция объекта изменяется по линейному закону, т.е. скорость постоянна на всем периоде анимации, что негативно сказывается на реалистичности.

В X3D для создания анимаций, лишенных вышеперечисленных недостатков, используются узлы-последователи (Followers). Они реализованы как линейные фильтры.

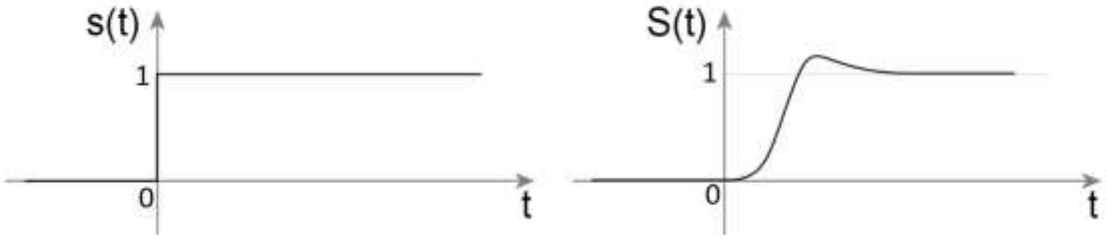
Для описания динамических систем применяется метод импульсной характеристики – реакции системы на дельта-функцию.

$$\delta(t) = \begin{cases} +\infty, t = 0 \\ 0, t \neq 0 \end{cases}$$

В зависимости от вида импульсной характеристики линейные фильтры подразделяются на фильтры с конечной импульсной характеристикой (КИХ-фильтры) и фильтры с бесконечной импульсной характеристикой (БИХ-фильтры). У КИХ-фильтров импульсная характеристика достигает нуля за конечный промежуток времени. У БИХ-фильтров импульсная характеристика никогда не возвращается в 0, но при этом может стремиться к нему асимптотически. КИХ-фильтрам в X3D соответствуют узлы-преследователи (Chaser), а БИХ-фильтрам – узлы-демпферы (Damper).

Другим методом описания динамических систем является их переходная характеристика, или реакция на ступенчатую функцию.

$$s(t) = \begin{cases} 0, t < 0 \\ 1, t \geq 0 \end{cases}$$



Далее при описании фильтров будем опираться на их переходную характеристику.

На линейные фильтры, используемые в последователях X3D, накладываются два ограничения:

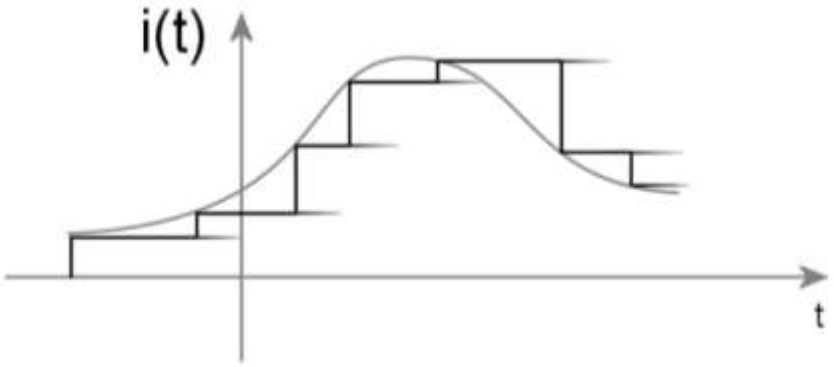
1) Фильтр должен быть каузальным. Иными словами, сигнал на выходе должен изменяться только после поступления входного сигнала, но не до.
2) Переходная характеристика фильтра должна приближаться к 1 при больших t.

Использование линейных фильтров в основе последователей позволяет добиться большей реалистичности анимаций за счет имитации инерции.

Пусть на вход фильтра подается непрерывный сигнал $i(t)$. Тогда выходным сигналом будет $o(t) = T(i(t))$.

Любой непрерывный сигнал может быть аппроксимирован суммой набора сдвинутых по времени и масштабированных ступенчатых функций.

$$i(t) = \sum_n a_n s(t - T_n)$$



Линейные фильтры обладают двумя свойствами, важными в данном контексте:

1) Инвариантность к сдвигу. Если подать на вход линейного фильтра тот же сигнал в более поздний момент времени, выходной сигнал будет точно таким же, как и ранее, но тоже сдвинутым по времени на ту же величину.

$$T(i(t + \Delta)) = (T(i))(t + \Delta)$$

2) Результатом суперпозиции нескольких сигналов, поданных на вход линейного фильтра, будет выходной сигнал, представляющий собой суперпозицию выходных сигналов для каждого из входных. Иными словами, если масштабировать входной сигнал, выходной сигнал также будет отмасштабирован соответствующим образом, а если подать на вход сумму двух сигналов, на выходе будет сумма ответов на них.

$$T(ai_1 + bi_2)(t) = aT(i_1(t)) + bT(i_2(t))$$

С учетом этого выходной сигнал линейных фильтров также может быть аппроксимирован суммой ступенчатых функций, каждая из которых является ответом на одну функцию из суперпозиции ступенчатых функций входного сигнала. Это позволяет

узлам-последователям формировать ответы на несколько идущих последовательно входных событий, как будет описано далее.

Описание общих полей последователей:

isActive – булев атрибут, активен последователь или нет.

initial_destination | - если эти поля заданы, последователь в начальный момент времени

initial_value | осуществляет анимацию от initial_value к initial_destination.

set_destination – входное событие, определяет конечное значение анимации. Начальным значением является значение на выходе узла в момент поступления события.

set_value – входное событие, позволяет напрямую передать на выход соответствующее значение, после чего осуществляется анимация от него к полученному ранее set_destination. Так, если необходимо запрограммировать анимацию от одного значения до другого, начальное может быть подано на set_value, а конечное – на set_destination.

value_changed – выходное событие.

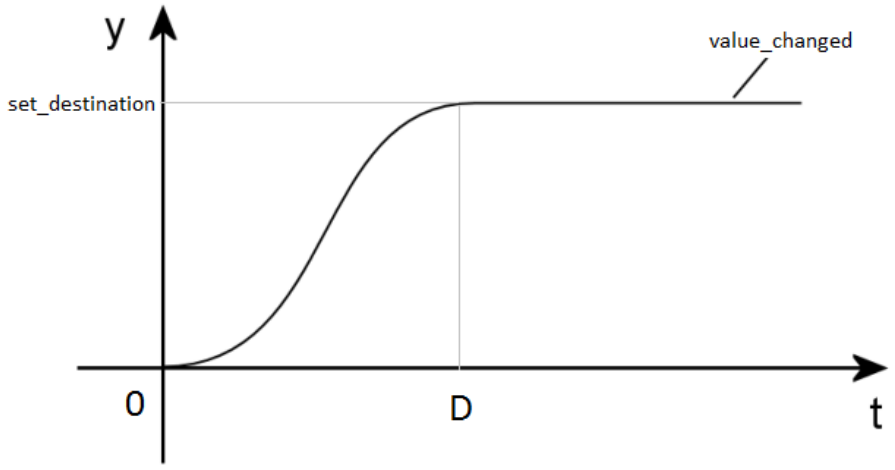
Преследователи (Chaser)

Преследователи реализованы на концепции КИХ-фильтров. Их переходная характеристика может быть описана следующим образом:

$$h(t) = \begin{cases} 0, t < 0 \\ \dfrac{1 - \cos \pi \dfrac{t}{D}}{2}, 0 \le t < D \\ 1, t \ge D \end{cases},$$

где D – интервал времени, за который выходной сигнал достигает уровня входного (в случае переходной характеристики - единицы).

При этом если на вход фильтра перестает поступать сигнал, то через интервал D сигнал на выходе также прекращается.



Описание полей:

duration – время в секундах, требуемое для завершения анимации (соответствует параметру фильтра D). Если значение duration равно 0, фильтр просто передает вход на выход без анимации.

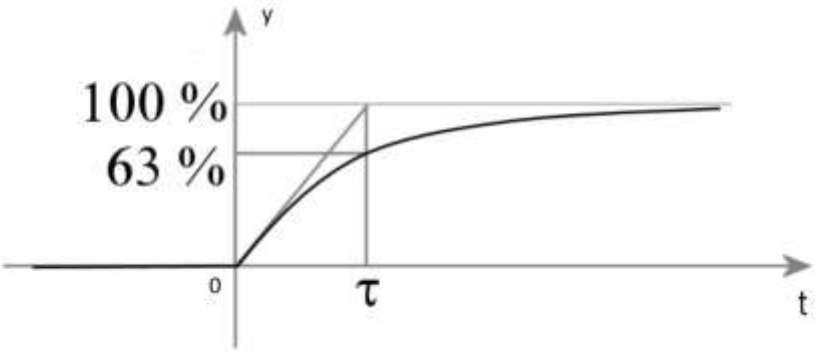
Демпферы (Damper)

Демпферы реализованы на концепции БИХ-фильтров. Их переходная характеристика может быть описана следующим образом:

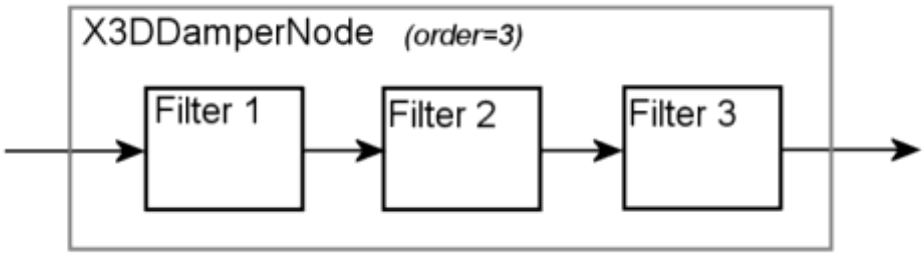
$$h(t) = \begin{cases} 0, t < 0 \\ 1 - e^{-\frac{t}{\tau}}, t \geq 0 \end{cases},$$

где τ - интервал времени, за который выходной сигнал достигает уровня $(1-\frac{1}{e})$ (переходная характеристика достигает значения 0,63).

Поскольку выходной сигнал асимптотически стремится к единице, анимация бесконечна. Поскольку это не имеет смысла в реальной реализации, в узлы X3D был включен дополнительный параметр ε - точность, по достижении которой генерирование сигнала прекращается и анимация заканчивается.



В X3D есть возможность последовательного подключения до 5 БИХ-фильтров. При этом выход предыдущего фильтра подается на вход следующего. Входом узла будет вход первого фильтра в цепочке, а выходом – выход последнего.



Количество подключенных последовательно фильтров называется порядком демпфера. Увеличение порядка делает анимацию более плавной, но также увеличивает задержку между входом и выходом.

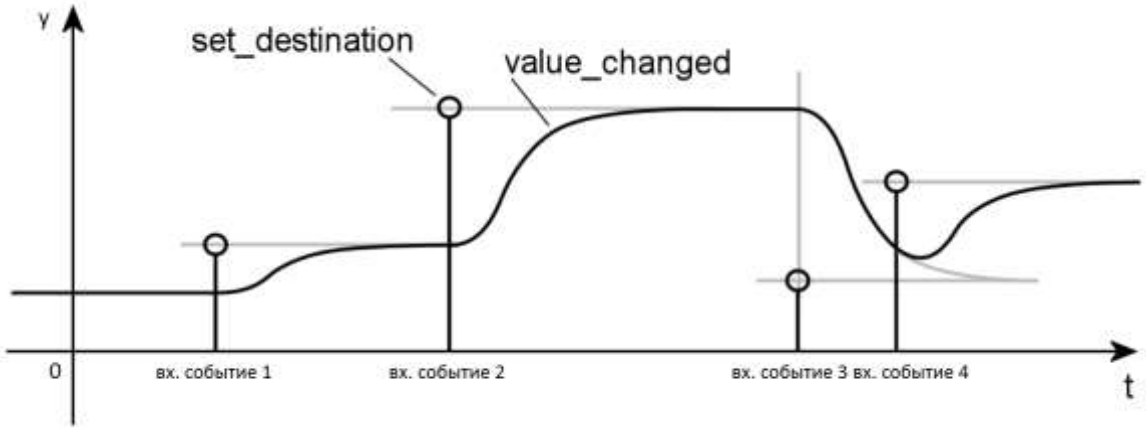
Описание полей:

tau – время в секундах, необходимое, чтобы анимация завершилась на 63% (соответствует параметру фильтра τ). Если значение tau равно 0, фильтр просто передает вход на выход без анимации.

eps – точность (ε), по приближении на которую к значению set_destination, формирование value_changed прекращается.

order – порядок демпфера (количество подключенных последовательно фильтров). Если значение order равно 0, фильтр просто передает вход на выход без анимации.

Важным свойством использования последователей является возможность реакции на поступление входного сигнала после начала анимации, но до ее завершения.



Благодаря вышеописанным свойствам линейных фильтров выходной сигнал может быть представлен в виде суммы сигналов, являющихся реакциями на ступенчатые функции, которые аппроксимируют входной сигнал произвольной формы. Это позволяет узлу корректно и естественно реагировать на входные сигналы, поступающие в процессе анимации.

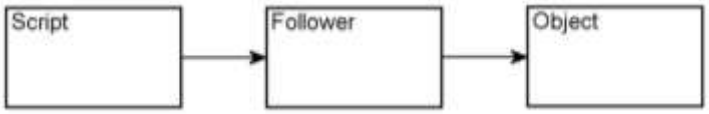
На рисунке изображена ситуация, когда на вход узлу Chaser последовательно подается 4 события. К моменту поступления события 1 узел находится в состоянии покоя (анимация не осуществляется). После поступления события 1 создается переход к новому значению value_changed, включающий в себя ускорение в начале и замедление в конце. По достижению конечного значения анимация останавливается до момента прихода события 2. Такая же ситуация наблюдается с наступлением события 3.

Входное событие 4 поступает на вход в момент осуществления перемещения к точке, заданной событием 3. В такой ситуации выходной сигнал формируется как суперпозиция ответов на входные события 3 и 4, что позволяет осуществить плавный отход от траектории реакции на событие 3 и перейти на траекторию ответа на событие 4.

Сценарии использования последователей

Создание анимации

Основная сфера применения последователей – создание анимации, параметры которой задаются во время выполнения. Когда объекту требуется переместиться, изменить цвет, форму и т.д., скрипт посылает сигнал узлу-последователю, который просчитывает анимацию от текущего значения величины к заданному. При этом поддерживается возможность получения сигналов в процессе анимации.

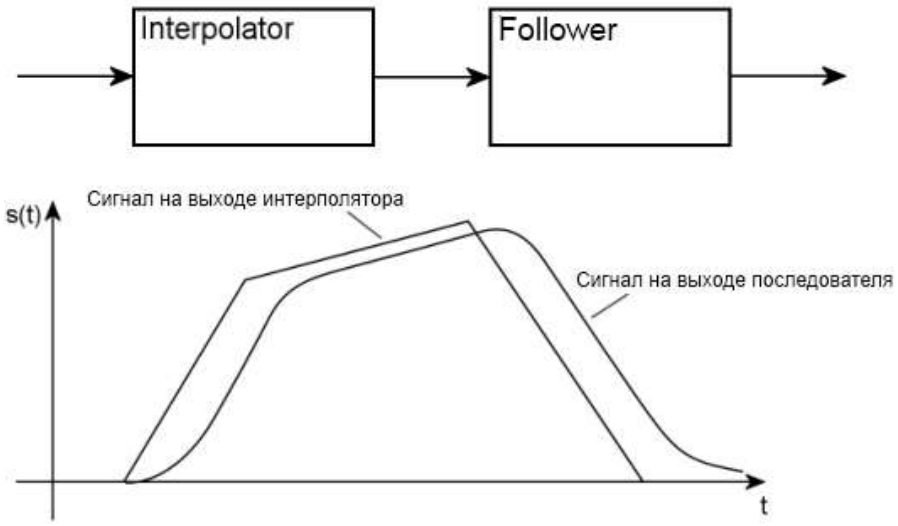


В примере скрипт указывает преследователю координату щелчка по поверхности, и преследователь вычисляет траекторию куба от его текущего местоположения к заданному.

```
<script>
  function move(point) {
    document.getElementById("BoxChaser").setAttribute("set_destination", point)};
  }
</script>
<X3D>
  <Scene>
    <PositionChaser DEF="BoxChaser" id="BoxChaser"
                    duration="1" initialDestination="-3 0 0">
    </PositionChaser>
    <Transform onClick="move(event.hitPnt);">
      <Shape>
        <Appearance>
          <Material></Material>
        </Appearance>
        <Box size="10 0.1 10"></Box>
      </Shape>
    </Transform>
    <Transform DEF="Box">
      <Shape>
        <Appearance>
          <Material></Material>
        </Appearance>
        <Box></Box>
      </Shape>
    </Transform>
    <ROUTE fromNode="BoxChaser" fromField="value_changed"
           toNode="Box" toField="set_translation">
    </ROUTE>
  </Scene>
</X3D>
```

Сглаживание существующих анимаций

Если на вход последователя set_destination подается непрерывный сигнал, он осуществляет его низкочастотную фильтрацию, сглаживая резкие изменения и неровности. В сочетании с узлами-интерполяторами это может быть использовано для создания плавных анимаций по заранее заданным траекториям с минимально необходимым количеством опорных точек.



Варьируя параметры последователя, можно добиться необходимого баланса между точностью анимации и ее гладкостью.

В примере преследователь сглаживает выходной сигнал интерполятора позиции для создания плавной анимации перемещения сферы.

```
<Scene>
  <TimeSensor DEF="Time" cycleInterval="5" loop="true" enabled="true">
  </TimeSensor>
  <PositionInterpolator DEF="PosInt" key="0 0.25 0.5 0.75 1"
                        keyValue="-3 -3 0, -3 3 0, 3 3 0, 3 -3 0, -3 -3 0">
  </PositionInterpolator>
  <PositionChaser DEF="PosCh" duration="1"></PositionChaser>
  <Transform DEF="Trans" translation="-3 -3 0">
    <Shape>
      <Appearance>
        <Material></Material>
      </Appearance>
      <Sphere></Sphere>
    </Shape>
  </Transform>
  <ROUTE fromNode="Time" fromField="fraction_changed"
         toNode="PosInt" toField="set_fraction">
  </ROUTE>
  <ROUTE fromNode="PosInt" fromField="value_changed"
         toNode="PosCh" toField="set_destination">
  </ROUTE>
  <ROUTE fromNode="PosCh" fromField="value_changed"
         toNode="Trans" toField="set_translation">
  </ROUTE>
</Scene>
```

Приложение 1

Типы данных полей

Название типа	**Описание**	**Примеры значений**
SFBool	Одиночное булево значение	`true`
MFBool	Несколько булевых значений	`true false false true`
SFColor	Одиночное значение цвета в формате RGB	`0 0.5 1.0`
MFColor	Несколько значений цвета в формате RGB	`1 0 0, 0 1 0, 0 0 1`
SFColorRGBA	Одиночное значение цвета в формате RGB с alpha-каналом (прозрачность)	`0 0.5 1.0 0.75`
MFColorRGBA	Несколько значений цвета в формате RGB с alpha-каналом (прозрачность)	`1 0 0 0.25, 0 1 0 0.5, 0 0 1 0.75`
SFInt32	Одиночное 32-битное целое число	`0`
MFInt32	Несколько 32-битных целых чисел	`1 2 3 4 5`
SFFloat	Одиночное число с плавающей точкой с одинарной точностью	`1.0`
MFFloat	Несколько чисел с плавающей точкой с одинарной точностью	`-1 2.0 3.14159`
SFDouble	Одиночное число с плавающей точкой с двойной точностью	`2.7128`
MFDouble	Несколько чисел с плавающей точкой с двойной точностью	`-1 2.0 3.14159`
SFImage	Одиночная картинка	`2 2 3 0xFF0000 0x00FF00 0x0000FF 0xFF0000`
MFImage	Несколько картинок	`2 2 3 0xFF0000 0x00FF00 0x0000FF 0xFF0000, 2 2 3 0x00FF00 0x0000FF 0xFF0000 0xFF0000`
SFNode	Одиночный узел	`<Shape></Shape>`
MFNode	Несколько узлов	`<Shape></Shape> <Transform></Transform>`
SFRotation	Одиночный массив вращения (3 значения вектора оси, угол в радианах)	`0 1 0 1.57`
MFRotation	Несколько массивов вращения	`0 1 0 0, 0 1 0 1.57, 0 1 0 3.14`
SFString	Одиночная строка	`"Hello world"`
MFString	Несколько строк	`"Hello" "world"`
SFTime	Одиночное значение времени	`0`
MFTime	Несколько значений времени	`–1 0 1 567890`
SFVec2F/SFVec2D	Одиночный вектор 2 чисел с плавающей точкой с одинарной/двойной точностью	`0 1.5`
MFVec2F/MFVec2D	Несколько векторов 2 чисел с плавающей точкой с одинарной/двойной точностью	`1 0, 2 2, 3 4, 5 5`
SFVec3F/SFVec3D	Одиночный вектор 3 чисел с плавающей точкой с одинарной/двойной точностью	`0 1.5 2`

MFVec3F/MFVec3D	Несколько векторов 3 чисел с плавающей точкой с одинарной/двойной точностью	10 20 30, 4.4 –5.5 6.6

Приложение 2

Параметры сцены X3DOM

Параметр	Значения	По умолч.	Описание
showLog	true, false	false	Скрыть/отобразить консоль с журналом
showStat	true, false	false	Скрыть/отобразить оверлей со статистикой
showProgress	true, false, bar	true	Скрыть/показать индикатор загрузки. По умолчанию используется вращающийся индикатор. Значение ‘bar’ выведет полосу загрузки.
width	Количество пикселей	300px	Ширина окна
height	Количество пикселей	150px	Высота окна

Приложение 3

Команды управления сценой X3DOM

Управление камерой в X3DOM осуществляется в одном из следующих режимов:

Изучение (Examine)
Активируется нажатием клавиши "e".

Функция	Кнопка мыши
Вращение	Лев / Лев + Shift
Панорамирование	Сред / Лев + Ctrl
Зум	Прав / Колесо / Лев + Alt
Установить центр вращения	Дв. лев. щелчок

Ходьба (Walk)
Активируется нажатием клавиши "w".

Функция	Кнопка мыши
Идти вперед	Лев
Идти назад	Прав

Полет (Fly)
Активируется нажатием клавиши "f".

Функция	Кнопка мыши
Двигаться вперед	Лев
Двигаться назад	Прав

Вертолет (Helicopter)
Активируется нажатием клавиши "h".
Смотреть вниз/вверх: клавиши 8/9
Подниматься/опускаться: клавиши 6/7.

Функция	Кнопка мыши
Двигаться вперед	Лев

Изучение (Look at)
Активируется нажатием клавиши "l".

Функция	Кнопка мыши
Приблизить	Лев
Отдалить	Прав

Игровой режим (Game)
Активируется нажатием клавиши "g".
Для вращения камеры двигайте мышью.

Функция	Клавиша
Двигаться вперед	Стрелка вверх

Функция	Клавиша
Двигаться назад	Стрелка вниз
Стрейфиться влево	Стрелка влево
Стрейфиться вправо	Стрелка вправо

Осмотр (Lookaround)
Активируется нажатием клавиши "o".
Для вращения камеры используются левая и правая кнопки мыши.

Во всех режимах доступны следующие команды управления камерой:

Функция	Клавиша
Сброс к начальному положению камеры	r
Показать всю сцену	a
Выпрямить камеру	u
Переключение к следующей точке наблюдения	PgDn
Переключение к предыдущей точке наблюдения	PgUp
Уменьшение скорости навигации	+
Увеличение скорости навигации	-

Для работы с использованием устройства сенсорного ввода наиболее подходящим является режим изучения (Examine).

Функция	Жест мультитача
Вращение вокруг произвольной точки	Сдвиг одним пальцем
Панорамирование	Сдвиг двумя пальцами
Вращение вокруг направления взгляда	Вращение двумя пальцами
Зум	Раздвинуть два пальца
Установить центр вращения	Двойное касание одним пальцем

Прочие клавиши управления сценой:

Функция	Клавиша
Переключение режимов обработки событий щелчка / касания	p
Переключение режимов визуализации (облако точек / каркасная модель / твердотельная модель)	m
Отобразить / скрыть консоль с журналом	d
Включение / отключение удаления мелких объектов (small feature culling)	s
Включение / отключение отсечения по пирамиде видимости (frustum culling)	c
Вывести на консоль информацию о текущих параметрах Viewpoint	v
Ориентировать камеру как первый объявленный ы сцене источник освещения	t