X3D (Extensible 3D)

При активной поддержке Web3D-консорциума в 1998 году была образована группа разработки X3D, целью которой было создание расширяемой трехмерной графической спецификации следующего поколения. X3D (Extensible 3D), по сути, является расширением языка VRML 97 с использованием возможностей расширяемого языка описания Web-страниц XML (Extensible Markup Language).

X3D рассматривается как преемник VRML. Методы моделирования и представления в VRML-формате с использованием расширений X3D применяются сегодня для представления сложного трехмерного Интернет-контента (вплоть до промышленных моделей систем автоматизированного проектирования CAD/CAM).

Для просмотра файлов мы используем свободно распространяемый просмотрщик - FreeWRL

Модели высокого качества и сложные формы создают большие файлы данных, которые трудно передавать по Сети.

Можно, конечно, пожертвовать точностью и подробностью представления и для повышения быстродействия уменьшить размер файла, но тогда мы лишимся большой части приложений.

Предложенная фирмой Lattice структура и новый XVL-формат (eXtensible Virtual world description Language), который описывает XML-расширение для X3D — это новая форма описания трехмерной модели, предназначенная для решения вышеупомянутых проблем.

Lattice-структура состоит, собственно, из Lattice Surface (поверхности из сплайновых «заплаток» — Gregory patch) или Lattice Mesh (полигональной каркасной сетки).

Использование Lattice-структуры и описывающего ее формата XVL как расширения X3D-стандарта для создания сложных трехмерных сцен в сети Интернет позволяет обходиться маленькими файлами, которые быстро передаются по Сети, но тем не менее сохраняют достаточную точность отображения и большое количество деталей в представляемых моделях.

Это достигается следующим образом: изначально моделируется произвольная сплайновая форма — Lattice-поверхность (очевидно, что даже само представление модели в виде сплайновой поверхности значительно экономичнее), а затем, уже после передачи данных на компьютер пользователя, Lattice-поверхности преобразуются браузером в полигональные сетки для последующей VRML-визуализации.

Между Lattice-поверхностью и Lattice-сеткой — взаимно однозначное соответствие (они имеют ту же самую топологию и структуру), и специальный алгоритм может быстро переводить одно в другое.

Каркасное представление Lattice-сетей обеспечивает пользователей простым в использовании и легко масштабируемым решением для отображения трехмерных моделей с поддержкой их целостности, а Lattice-поверхность гарантирует гладкость и высокую достоверность при отображении мелких деталей.

При этом использование Lattice-структуры как открытого расширения X3D уже поддержано целым рядом производителей, которые стремятся получить быстрый и эффективный способ для передачи и представления высококачественной 3D-графики в сети Интернет.

Итак, Lattice Mesh позволяет создавать маленькие файлы, а Lattice Surface представляет точные поверхности. Формат XVL может описывать и Lattice Surface, и Lattice Mesh, облегчая эффективную передачу данных и высокую точность представления.

X3D – описывается с помощью XML (язык разметки гипертекста)

XML — текстовый формат, предназначенный для хранения структурированных данных для обмена информацией между программами, а также для создания на его основе более специализированных языков разметки.

Тег - элемент языка разметки гипертекста. В XML тег является элементом документа, а текст, содержащийся между начальным и конечным тегом — содержанием элемента.

Язык схем DTD (DTD schema language) — искусственный язык, который используется для записи фактических синтаксических правил метаязыков разметки текста XML

Создадим на рабочем столе (или в любом другом удобном для вас месте) папку и назовем ее «X3D», в ней будут храниться все наши примеры.

Теперь в ней создадим файл с расширением .x3d (либо сперва создайте файл .txt, а затем сохраните его с нужным расширением).

Откройте файл через редактор и в начале документа пропишите заголовок:

```
<?xmlversion="1.0" encoding="UTF-8"?>
```

<?....?> - говорит интерпретатору, что это заголовок
xml version=”1.0” – версия действующего xml
encoding – указание кодировки, в X3D используется кодировка UTF-8.

Далее запишем, в нашем коде, следующее:

```
<X3D version='3.2'>
        - основная часть сцены
</X3D> - символ « /»
```

(слэш) обозначает, что тег закрыт

После того, как мы с вами написали всю необходимую информацию для интерпретатора, пришло время объявить и саму сцену.

<Scene>... </Scene>

Между тегами <Scene> располагаются все объекты, сенсоры, скрипты и все необходимое для получения конечного результата.

<!...> - указание для интерпретатора, что это элемент DTD.
<!--..... --> - комментарий

Простейшие графические объекты:

- Box (прямоугольный параллелепипед),
- Cylinder (цилиндр),
- Cone (конус),
- Sphere (сфера).

Узлы помещаются в узел Shape, и не могут быть использованы в качестве корневых или узлов-предков, т.е. они являются листьями в графе X3D-сцены.

Эти объекты имеют атрибут (свойство) solid (признак проницаемости) со значением по умолчанию true (сплошной).

Узел Shape содержит один объект, например, Box, Cone, Cylinder, Sphere, ElevationGrid, Extrusion, IndexedFaceSet, IndexLineSet, PointSet, Text.

Узел <Shape> кроме узла содержит в себе узел <Appearance> (внешность), содержащий информацию о внешнем виде этого узла.

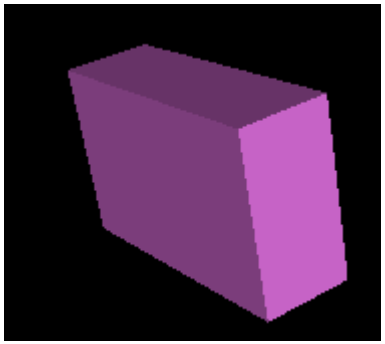
Эта информация находится в узлах Material, ImageTexture и других потомках узла Appearance.

Такое вложение позволяет повторно использовать описания внешних свойств объектов с помощью полей DEF и USE. Это бывает полезным при наличии множества объектов с одинаковым внешним видом. При этом ускоряется рендеринг.

Узлы типа <Appearance> позволяют создавать разнообразные свойства поверхности, например текстуру "под дерево", которые затем могут применяться к различным объектам или поверхностям создаваемого мира.

Узел <Box> – прямоугольный параллелепипед (ящик) с центром в начале связанной системы координат (ССК), стороны которого параллельны координатным осям. По умолчанию размер каждого ребра параллелепипеда равен 2 м, т.е. их координаты в осях ССК изменяются от -1 до +1.

В поле size задаются размеры параллелепипеда по осям X, Y, Z ССК, например, <Box size='1 2 3'/>. Все три числа должны быть положительными. Браузер отображает только внешние грани параллелепипеда (вид параллелепипеда изнутри не определен).



box2.X3D

```
<x3d>
<Scene>

<Shape>
    <Box size='1 2 3'/>
    <Appearance>
        <Material diffuseColor='1  0.5  1'  />
    </Appearance>

</Shape>

</Scene>
```

Узел <Cylinder> – цилиндр имеющий по умолчанию высота 2 м и радиус основания 1 м. Начало ССК находится в геометрическом центре.

Ось параллельна вертикали. Для того чтобы задать свои размеры, нужно ввести желаемые значения в поля radius (радиус) и height (высота). Эти числа должны быть неотрицательными.

В этом узле есть еще поля side (признак видимости боковой стороны), top (признак видимости верхнего основания) и bottom (признак видимости нижнего основания). В эти поля можно поместить логические значения true или false.

С их помощью браузер определит, нужно ли отображать соответ-ствующую поверхность цилиндра. По умолчанию они имеют значение true, т.е. по умолчанию все стороны цилиндра будут отображены. Если концы цилиндра закрыты другими объектами, то их отображение лучше отключить, чтобы не задавать лишней работы браузеру.

Следующий код: <Cylinder radius='2' bottom='false' top='false'/>
размечает цилиндр с высотой (по умолчанию) и радиусом оснований 2 метра, которые не видны.

Конус (Cone) сходен с цилиндром, только поле radius (радиус) заменено полем bottomRadius (радиус нижнего основания) и отсутствует поле top содержащее признак видимости верхнего основания.

В следующем коде все свойства можно опустить, т.к. их значения используются по умолчании

```
<Cone bottom='true' bottomRadius='1' height='1' side='true'/>
```

Центром конуса считается средняя точка на его оси. Итак, конус с полями по умолчанию будет простираться на 1 м ниже горизонтальной плоскости его ССК и на 1 м выше, и радиус основания тоже будет равен единице.

Ось конуса совпадает с осью Y его ССК.

Узел Sphere в качестве центра имеет точку с координатами (0,0,0) в ССК. Поле radius содержит числовое значение радиуса сферы и должно быть больше нуля, например,

```
<Sphere radius="7"/>
```

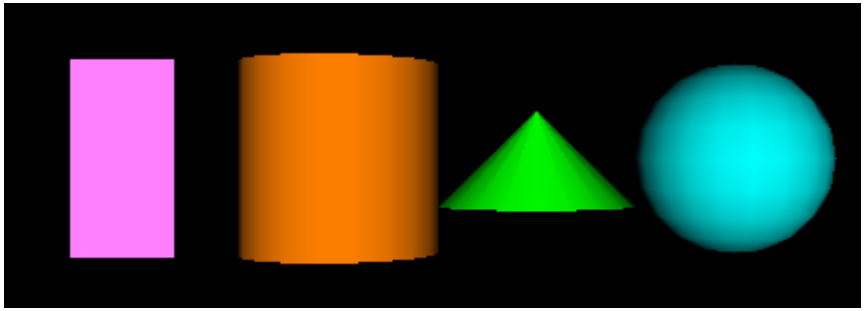
Пример:

```
<Scene>
    <Transform rotation='0 1 0 3'>
    <Shape>
        <Sphere/>
        <Appearance>
              <Material />
        </Appearance>
    </Shape>
      </Transform>

     <Transform translation='0 -2 0'>
            <Shape>
              <Text string='"Hello" "world!"' solid='false'>
                    <FontStyle justify='"MIDDLE" "MIDDLE"'/>
              </Text>
              <Appearance>
                    <Material />
              </Appearance>
           </Shape>
     </Transform>
</Scene>
```

hellow.X3D

Узел *<Transform rotation='0 1 0 3'>* *<Transform translation='0 -2 0'>*, позволяет менять положение следующего за ним объекта.



box3.X3D

Основы анимации

Поля узлов и их события можно связывать между собой. Способ связывания называется маршрутизацией (ROUTE).

Каждому из входящих в маршрут узлов нужно дать уникальное имя в поле DEF.

Например, чтобы связать между собой ис-ходящее событие touchTime узла с DEF='**SENSOR**' и входящее поле startTime узла с DEF='**SOUND**' (для запуска звука после щелчка мышкой по сенсору), необходимо маршрутизировать эти события следующим образом

*<ROUTE fromField='touchTime' fromNode='**SENSOR**' toField='s**tartTime**' toNode='**SOUND**'/>*

Если при помощи USE создать несколько объектов с одним именем и внести их имена в маршрут, происходящие события коснутся каждого из них. Если нужно, чтобы в событии участвовал только один экземпляр объекта, дайте ему уникальное имя. Или же воспользуйтесь декларацией PROTO.

Маршрут соединяет узел, в котором происходит событие, с узлом, который должен получить сообщение об этом событии.

Маршрут — это синтаксическая конструкция (невидимый узел), которая описывает путь перемещения информации между полями разных узлов. Маршруты отвечают за создание связей между узлами. Марш-руты могут быть помещены в конец X3D-файла или включены в описание прототипа.

Если узел-источник посылает дополнительно информацию о новых событиях, то такая совокупность сообщений называется цепочкой событий. Время появления всех сообщений в цепочке событий определяется по самому первому событию.

Многие узлы имеют поля событий двух типов: eventIn и eventOut.

Исходящие события (eventOut) отсылают информацию, например, новое значение какой-то величины или время щелчка мышью.

Входящие события (eventIn) принимают информацию извне узла и обрабатывают её.

Каждое событие имеет ассоциированный с ним тип данных.

Некоторые узлы содержат поля, описываемые как exposed (открытые).

Это означает, что в узле есть два события для этого поля: set_имяПоля и имяПоля_changed.

При помощи первого из них можно устанавливать значение поля, а при помощи второго — извещать окружающий мир о том, что значение поля изменилось.

Если вы используете set_имяПоля для того, чтобы изменить значение поля, то узел затем породит событие имяПоля_changed.

Для простоты использования части set_ и _changed можно опускать, а браузер сам вычислит, какое именно событие вы имели в виду.

Если поле не является открытым (exposed), то его значение не может быть изменено при помощи событий и всегда совпадает со значением, заданным в файле.

Графические объекты могут реагировать на действия пользователя (или скриптов) и обеспечивать обратную связь. Управление событиями и порядком, в котором они происходят достигается путем создания связей между узлами, т.е. дорожек, по которым передаются сообщения от узла к узлу. Эти связи невидимы, потоки событий не имеют явного визуального представления, но связи пронизывают собой всю модель.

Входящие события подобны приемникам, которые прислушиваются к сообщениям и присматриваются к событиям из внешнего по отношению к данному узлу мира и принимают их на обработку.

Исходящие события подобны передатчикам, посылающим сообщения от узла к другим узлам.

Открытые поля объединяют обе эти функции. Открытое поле по имени nam имеет входящее событие set_nam и исходящее событие nam_changed.

Обычно части set_ и _changed в имени события опускаются, достаточно указать имя поля, а браузер догадается, что вы имели в виду.

Нужно, чтобы события кто-то принял. Достигается это при помощи связывания узлов маршрутами (routes). Они как дорожки, по которым информация продвигается от исходящего события одного узла ко входящему событию другого.

Можно соединить входящие события нескольких узлов с исходящим событием одного узла. Тогда одно исходящее событие повлечет за собой целый каскад других событий. Это называется расходящимся веером событий.

Сходящиеся веера, т.е. ситуации, когда несколько исхо-дящих событий связаны с одним входящим, теоретически тоже допустимы.

Если два собы-тия войдут в одни узел в одно время, то результат предсказать невозможно. Одновременность входящих в один узел событий может являться лишь ре-зультатом работы вашей программы, поэтому следует аккуратно отслеживать маршруты.

Принято записывать узлы ROUTE в конце X3D-файла. Они не входят в состав никаких узлов и абсолютно независимы от тех узлов, которые связывают. Нужно отметить, что маршруты могут связывать между собой только поля одного и того же типа. Например, нельзя связать поле типа SFTime с полем типа SFBool. Если такое совсем уж необходимо, нужно будет написать сценарий, преобразующий значение одного типа в другой тип.

События – это потоки информа-ции состоящие из двух частей: сообщения (значения определенного типа) и момента времени t.

Значение сообщения может быть любого типа, например SFTime или SFString.

Если узел в состоя-нии это событие обработать, вы можете его послать.

Моментом t распоряжается браузер и нам он абсолютно неподвластен. Абсолютное значение t особой роли не играет, важен лишь порядок событий: что произошло раньше, а что позже.

Как правило, события обрабатываются в порядке возрастания значений t. Когда порождается одно событие, это вызывает к жизни еще одно, уже в другом узле, и т.д. — такая цепочка называется каскадом событий.

Все события внутри каскада имеют одно значение tc точки зрения браузера. В случае сходящегося веера событий, если два входящих собы-тия будут иметь одно t результат будет неопре-деленным.

Внутри каскадов могут встретиться **циклы** — это когда одно из событий инициирует другое, а то, в свою очередь, запускает первое. Если цикл и присутствует, он не будет крутиться вечно, а будет исполнен всего один раз.

Начальные события, т.е. события, не порожденные другими события-ми в каскаде, могут порождаться **узлами-сенсорами** и **узлами-сценариями**.

Узлы могут порождать события только в ответ на какие-то другие события. Поэтому **сенсоры** и **сценарии** являются ключевыми точками в управлении анимацией и интерактивностью в **X3D**.

Узлы же, отвечающие за собственно анимацию с технической точки зрения, – это **интерполяционные узлы** и **узлы-переключатели**, а также открытые поля всех типов узлов. Между узлами этих двух типов, т.е. триггеров анимации и ее исполнителей, могут находиться дополнительные **узлы-сенсоры** и **узлы-сценарии**, т.е. то что создаёт двигатель анимации.

Принимая сообщения, узлы могут изменять свое состояние.

Большинст-во узлов приспособлены для обработки поступающей к ним информации, называемой входящими событиями (incoming event — eventIn).

Принимая eventIn - события, узел изменяет свое состояние. Например, цвет узла может быть изменен при помощи входящего события se_tcolor.

Об изменении своего состояния узел может сообщить при помощи _changed - события (они относятся к исходящим событиям — evenOut - событиям).

С помощью eventOut - событий можно отправлять другим узлам сообщения с предупреждениями об изменениях в узле-источнике.

Color_changed, position_changed — примеры событий, распространяющих сведения об изменении состояния узла.

Сенсоры

В X3D-анимации используются три основных класса узлов:

- сенсоры,
- интерполяторы
- сценарии.

Сенсоры окружающей среды не принимают ввода данных непосредственно от пользователя, но вместо этого наблюдают процессы окружающей среды, такие, как течение време-ни и расположение пользователя.

Узлы-датчики

Узлы типа Sensor (датчик) позволяют динамически порождать события в X3D-мирах.

Для порождения событий вследствие действий пользователя применяются датчики:

- ProximitySensor,
- VisibilitySensor,
- TouchSensor,
- SphereSensor,
- PlaneSensor,
- CylinderSensor,
- группирующий узел Collision.
- Узел TimeSensor порождает события связанные с течением времени.

Можно сделать так, чтобы датчики зависели друг от друга, т.е. при отключении одних узлов активизировались другие.

После того как датчик (или сценарий) генерирует начальное событие, оно распространяется по заданным маршрутам к другим узлам. Последние, в свою очередь, могут генерировать собственные события. Такой процесс называется каскадом событий (event cascade). Каскадным событиям присваивается одно и то же время появления, т.е. считается, что все события происходят одновременно. Более того, некоторые датчики могут генерировать сразу несколько событий; таким образом, может возникнуть сразу несколько каскадов событий.

Узлы Sensor могут быть только дочерними узлами. Для них в качестве родительских выступают группирующие узлы. Некоторые Sensor-ы являются датчиками для устройств-манипуляторов (pointing device sensors), т.е. они активизируются при воздействии указателя мыши или в результате действий браузера.

Датчик-манипулятор начинает действовать после того, как пользователь, например, при помощи мыши воздействует на олбъект связанный с датчиком (имеющий с датчиком общего родителя).

Датчик воздействует на все объекты родительской группы датчика (исключения составляют датчики-якоря, которые сами являются группирующими узлами).

Ниже приведен список узлов-манипуляторов:

- Anchor (якорь);
- TouchSensor (контактный датчик);
- CylinderSensor (цилиндрический датчик);
- PlaneSensor (плоский датчик);
- SphereSensor (сферический датчик).

Последние три образуют подмножество датчиков перемещений (Drag Sensors).

С помощью датчика CylinderSensor можно генерировать события в том случае, если мышь (или другой манипулятор) активизирована и при этом перемещается, а в момент активизации ее указатель находился на одном из объектов родительской группы.

Если манипулятор активизирован, генерируется событие isActive=”TRUE”.

Рассмотрим четыре сенсора, которые мо-гут принимать информацию от пользователя реагируя на щелчки мышкой, буксировку при помощи мыши с нажатой кнопкой и т.д.

TouchSensor (датчик касания) реагирует на взаимодействие мыши с графическими объектами (потомка-ми своего родительского узла).

Когда мышь заходит в область этих объектов, генерируется событие isOver с па-раметром (с значением) TRUE, а когда выходит из нее – поле (событие) isOver с значением FALSE. Пока мышь пребывает в состоянии isOver=”TRUE”, то щелчок по любой её кнопке создаёт событие isActive=”TRUE” в тот момент, когда кнопка отпускается.

Событие isOver позволяет легко воплощать эффект перекатывания (rollover), т.е. технику, при которой при приближении мыши активные элементы окна меняют окраску или форму. Таким образом пользователь узнает, что перед ним - активный элемент.

Исходящее событие touchTime=”абсолютное время” посылается, когда мышью щелкают на объекте, сгруппированных с узлом TouchSensor. Если курсор уходит из области геометрии до того, как будет отпущена кнопка мыши, событие не посылается. Если же кнопка мыши отпущена, посылается событие со значением времени, когда кнопка была отпущена.

Исходящее событие touchTime генерируется, если одновременно будут выполнены следующие три условия.

Во-первых, указатель манипулятора располагается на геометрическом объекте, на котором он был изначально активизирован (поле isActive имеет значение TRUE).

Во-вторых, манипулятор переведен в неактивное состояние (сгенерировано событие isActive со значением FALSE).

В третьих, манипулятор располагается на объекте, рассмотренном в первых двух пунктах (поле isOver содержит значение TRUE).

SphereSensor (сферический датчик) - позволяет вращать объект вокруг связанной системы координат (ССК).

Датчик определяет, была ли нажата кнопка мыши, используя информацию геометрических объектов родительского узла. Это один из датчиков буксировки, т.к. их можно "перетаскивать" мышью в разных на-правлениях (вращать и перемещать) разными способами.

Этот сенсор, как и все, улавливает щелчки мышью на объектах братьях, т.е. потомках его родительского узла. Затем пользователь может его буксировать, не отпуская кнопки мыши. Это вызовет вращение геометрического объекта. Если autoOffset имеет значение TRUE, то враще-ние будет происходить между точками входа и выхода касания с геометрией сенсора.

Исходной позицией каждого нового вращения будет заключительная позиция предыдущего. Если там стоит FALSE, то каждый раз, когда пользователь касается геометрии, параметр вращения будет восста-навливать значение 0.

Если сенсор используется с параметром autoOffset=”TRUE”, то в момент освобождения кнопки мыши значение поля offset становится равным значению текущего поворота. Когда пользователь в следующий раз нажмет кнопку мыши на одном из объектов сенсора, параметры предыдущего поворота будут сохранены. Если же autoOffset=”FALSE”, то значение поля offset не изменяется. А в этом поле хранятся параметры вращения, на которое будет смещен сенсор с самого начала, как только произойдет нажатие на кнопку мыши.

Поля enabled (включен), isActive работают так же, как и в узле TouchSensor. Поле trackPointchanged содержит неза-крепленную свободную позицию буксировки на поверхности виртуальной сферы вращения. Поле rotation_changed передает значения приращения угла поворота ГО путём маршрутизации на поле rotation узла Transform.

Используя TimeSensor (датчик времени) можно генерировать события по мере того, как течет время в виртуальном мире. С помощью TimeSensor можно обеспечивать выполнение каких-либо действий через определенные интервалы времени или оповещать о наступлении некоторых событий.

CylinderSensor (цилиндрический сенсор) может вращаться во-круг своей оси Y.

Если вам захочется повращать цилиндрический сенсор вокруг оси Z, придется предварительно изменить положение всего сенсора при помощи объемлющего узла Transform.

CylinderSensor имеет поля diskAngle, maxAngle и minAngle.

Вращать цилиндр можно двумя способами.

- Сбоку, зацепив мышью бок цилиндра и "тащить" его поперек экрана. Чем дальше тянуть, тем больше он будет поворачиваться, и так до края.
- Если смотреть на цилиндр сверху, то выбрать точку и буксировать ее по окружности, вызывая тем самым вращение.

Всё зависит от поля diskAngle. Если вы смотрите на цилиндр так, что угол между вами и осью Y меньше, чем diskAngle, то он будет вести себя как диск и вращаться при движении мы-шью по окружности. В противном случае для того, чтобы цилиндр повернулся, вам нужно тащить его за бок мышью поперек экрана.

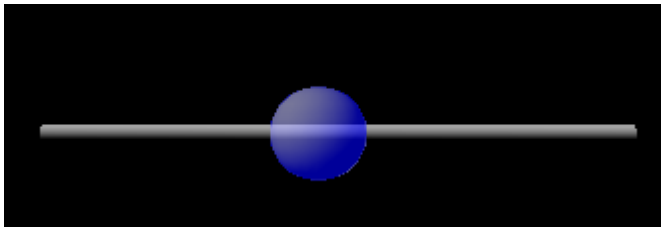
MaxAngle и minAngle используются для того, чтобы ограничить угол поворота цилиндра каким-то числом радиан.

Если minAngle больше, чем maxAngle (а такие значения предполагаются по умолчанию), сенсор имеет право повернуться на любой угол.

Если это не так, то поворот сенсора ограничен значениями из этих полей. Эта опция может быть использована, например, при создании виртуальных контроллеров громкости звука.

PlaneSensor (плоский сенсор) отслеживает перемещения мыши (или любого другого манипулятора) и преобразует координаты положения указателя мыши на экране в координаты своего локального двухмерного пространства.

Сенсор буксировки PlaneSensor можно перемещать мышкой параллельно некоторой плоскости. А именно он может передвигаться в направлении осей X и Y своей локальной системы координат. Локальная же координата Z фиксируется равной нулю.



Plane_sensor.X3D

Здесь, как и для остальных сенсоров буксировки, мы имеем события isActive и trackPoint_changed и поля enabled, autoOffset и offset.

Когда сенсор перемещается, его новое положение становится параметром исходящего события translation_changed. Поля maxPosition и minPosition ограничивают передвижение сенсора в каждом из двух направлений. В каждом из них хранится двумерный вектор (SFVec2f).

Если по какой-то из осей минимум и максимум совпадают, то сенсор не может перемещаться по этой оси. Если же минимум больше максимума (что под-разумевается по умолчанию), то сенсор не имеет ограничений по этой оси и может перемещаться как угодно далеко. В оставшемся случае перемещение ограничено минимумом и максимумом. В примере передвижение бегунка-шарика ограничено осью X путем остановки минимума и максимума по оси Y в нуль. Кроме того, бегунок не съезжает со стержня из-за того, что его minPosition и maxPosition по оси X совпадают с концами стержня.

С помощью inOut - поля enabled можно запретить или разрешить использование PlaneSensor. Когда значение этого поля установлено в TRUE, датчик соответствующим образом реагирует на действия пользователя. Если же значение равно FALSE, то датчик никак не реагирует на манипуляции пользователя и не генерирует исходящие события.

Если в поле enabled поступит значение FALSE, а поле isActive при этом содержит значение TRUE, то датчик становится и недоступным, и неактивным. Кроме того, генерируется исходящее событие isActive со значением FALSE. Датчик становится доступным и способным активизироваться пользователем при получении входящего события с указанием о переводе поля enabled в состояние TRUE.

Сенсор времени

Сенсор времени (TimeSensor) тикает и тикает снабжая X3D-мир сигналами точного времени, например, для анимации узлов, или для того, чтобы порождать регулярные или разовые события или управлять узлами-интерполяторами.

Его поле cycleInterval (интервал цикла) хранит величину временного интервала, который пройдет до перезапуска таймера.
Поле enabled (включен) используется для запуска и остановки таймера.
В поле loop (цикл) находится логическое значение, определяющее, отработает ли таймер ровно один раз или будет перезапускаться вновь и вновь, генерируя события каждые cycleInterval секунд. В полях startTime и stopTime находятся значения типа SFTime, указывающие время запуска и остановки таймера.

Исходящее событие cycleTime узла TimeSensor порождается всякий раз, когда таймер достигает своего интервала цикла, независимо от того, какое значение стоит в поле loop. Передавае-мым значением служит текущее время.

Итак, если у вас есть работающий по циклу TimeSensor с интервалом цикла в 1 с, событие cycleTime будет происходить каждую секунду и оно будет посылать значение, равное текущему времени, т.е. увеличивающееся каждый раз на единицу. Эта функция полезна как для регулярных повторяющихся событий, так и для однократных сигналов.

Для управления непрерывными анимациями нам нужен непрерывный же поток сигналов. Его можно создать при помощи события fractionChanged. Оно генерирует события с максимальной частотой, на ко-торую способен компьютер.

Спецификация не содержит указаний на то, как часто эти события будут происходить. Посылаемое значение относится к типу SFFloat и обозначает ту долю интервала цикла, которая реально прошла к настоящему времени.

Например, если интервал цикла установлен в 5 с, то событие fractionChanged, сгенерированное через 0.1 с, будет иметь значение 0.02.

В момент startTime значение fraction_changed равно нулю. Далее оно изменяется периодическим образом в интервале (0, 1).

В точках вида startTime+n*cycleInterval, где n — целое число, значение fractionChanged равно единице. Это событие используется для управления интерполяторами, связывая ключевые значения интерполируемых величин с определенными моментами времени.

Событие time генерируется одновременно с событием fractionChanged, но содержит абсолютное значение времени этого события. Событие isActive генерируется всякий раз, когда таймер запускается или останавли-вается. Оно несет в себе логическое значение TRUE или FALSE в зависимости от того, заработал таймер или остановился.

В следующем примере событие cycleTime циклического сенсора времени связано с открытым полем startTime нециклического узла AudioClip.

X3D-мир состоит только из текста и звука, который запускается каждые 2 с.

```
<TimeSensor DEF="MYTIMER"   cycleInterval="2"  loop="true"/>

<Sound maxBack="50" maxFront="50" minBack="10" minFront="10">

<AudioClip DEF="MYSOUND" pitch="0.5" url="Chimes.wav"/>

    </Sound>
      <Shape>
          <Appearance> <Material diffuseColor="1 0 0"/> </Appearance>
        <Text string="Now playing..."/>
      </Shape>

<ROUTE fromNode="MYTIMER" fromField="cycleTime"  toNode="MYSOUND" toField="startTime"/>
```

Интерполяторы

Узлы-интерполяторы (их шесть) играют важную роль в создании анимаций т.к. они меняют некоторую величину с течением времени.

Узлы-интерполяторы позволяют задавать ключевые кадры анимации и предоставляют браузеру позаботиться обо всем остальном. Таким образом каркас анимации задается сжато и просто. Нужно только помнить что для того, чтобы анимация текла плавно, без рывков, в нуле и единице поле keyValue должно иметь одно и то же значение.

Используя Interpolator можно задать кусочно-ли-нейную функцию f(t) на интервале от минус бесконечности до плюс беско-нечности.

Узлы данного типа созданы для организации линейной анимации по ключевым кадрам (linear keyframe animation).

Механизмы их действия похожи.

Механизм действия

Итак, интерполяторы применяются там, где нужно менять какую-то величину с течением времени. Они принимают сигналы времени с узла TimeSensor или подобного ему и производят линейную интерполяцию между соседними значениями (ключевыми значениями).

Для каждого ключе-вого значения существует ключ (key), дробь в интервале от 0 до 1.

Таймер генерирует события fraction_changed со скоростью, на ка-кую только способен. Это событие может быть маршрутизировано на вхо-дящее событие set_fraction интерполятора для того, чтобы установить пра-вильную точку цикла. Так, если вы маршрутизируете на интерполятор таймер с интервалом цикла, равным 10 с., интерполятор будет проходить по циклу свои ключевые значения каждые 10 с.

Всякий раз, когда интерполятор получает входящее событие, он гене-рирует исходящее событие с параметром, равным интерполированному значению величины. Оно, в свою очередь, может быть маршрутизировано на любое поле подходящего типа.

Интерполяторы могут быть помещены в любое место сцены, будь то корень или узел Transform. Поскольку они являются узлами, отвечающими только за "моторчик" интерполяции, то их конкретное место в иерархии сцены значения не имеет.

Имена полей у всех интерполяторов одинаковы. Разница между ними заключается лишь в том, что в полях keyValue (ключевое значение) и исходящих событиях value_changed находятся разные типы данных.

Итак, сначала идет входящее событие set_fraction. Оно получает события от таймера. Это дает информацию о текущем моменте цикла анимации и указывает, как далеко мы зашли в своем интерполяционном процессе. Следующее поле key (ключ) – это последовательность значений от 0 до 1, соответствующих ключевым точкам анимации.

Если вы хотите, чтобы в нулевой момент времени, в середине и в конце объект занимал опре-деленное положение, то вы создадите поле key со значением [0, 0.5, 1].

Каждое значение в поле key должно иметь соответствующее ему значение в поле keyValue, так что, если следовать приведенному выше примеру и использовать ScalarInterpolator, который интерполирует величины типа SFFloat, мы можем создать поле keyValue [0.0, 10.0, 0.0]. В момент времени 0 исходящее значение будет 0.0, в середине цикла — 10.0, а в 1 (конец цикла) - опять 0.0.

Между этими моментами времени значения будут интерполироваться линейно, т.е. в момент 0.25 цикла посылаемое значение будет равно 4. Значения передаются в качестве параметра исходящего события value_changed всякий раз, когда интерполятор получает событие fraction_changed.

Существует шесть типов таких узлов:

- ColorInterpolator (Интерполятор света);
- CoordinateInterpolator (Интерполятор координат);
- NormalInterpolator;
- OrientationInterpolator (Интерполятор ориентации);
- PositionInterpolator (Интерполятор положения);
- ScalarInterpolator (Скалярный интерполятор).

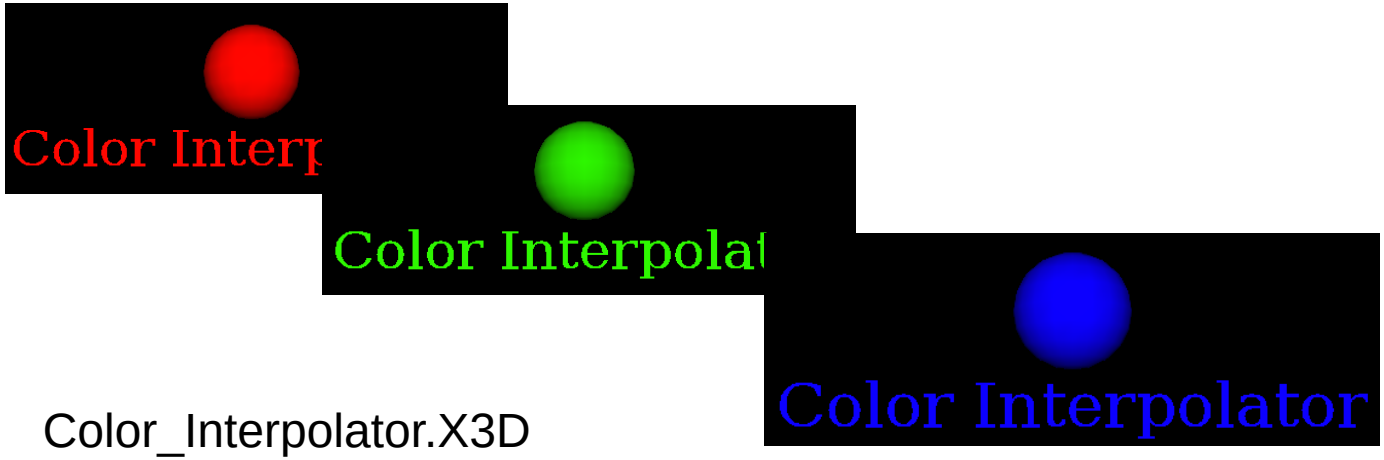
Интерполятор цвета ColorInterpolator

ColorInterpolator (интерполятор цвета) интерполирует RGB-тройку цветов.

Цвета декларируются в поле keyValue точно так же, как и в любом поле типа MFColor.

Если мы хотим, чтобы цвет изменялся от ярко-зеленого через синий к красному, то напишем следующее: key =”0,0.5,1” keyValue =”010, 001,100”

Заметьте, что здесь все-таки будет скачок цвета в конце интервала цикла. В примере цвет сферы и текста под ней равномерно изменяется под воздействием таймера.



Color_Interpolator.X3D

Интерполятор координат CoordinateInterpolator

Узел CoordinateInterpolator (интерполятор координат) слегка отличается от других.

Вместо того чтобы испускать одно значение при каждом щелчке таймера, он может посы-лать целое множество.

Например, целый набор IndexedFaceSet.

Отметим, что число значений координат в поле keyValue должно быть строго кратно числу ключей в поле key.

Коэффициентом же будет служить число координат, посылаемых в один заход.

Следует проявить аккуратность и поддерживать координаты в нужном порядке в каждом наборе.

Интерполятор ориентации OrientationInterpolator

OrientationInterpolator (интерполятор ориентации) производит сглаживание между значениями поворота (ось + угол) и может быть маршрутизирован на событие set_rotation узла Transform.

Таким образом, мы уже можем создавать анимированные точки обзора.

Если мы поместим точку обзора внутрь узла Transform, а сам узел Transform заставим изменять угол поворота, путешественник будет вращаться вокруг своей оси, не меняя точки, на которой он стоит.

В примере сохраним одну статичную точку обзора (назовем ее Entry) и добавим еще одну, заключенную в узел Transform.

Маршрутизируем узел OrientationInterpolator на со-бытие set_rotation объемлющего узла Transform второй точки наблюдения и запустим наш пример. Заметим, что меню точек обзора нашего мира содержит две точки. Сразу после загрузки мира вращаться еще ничего не будет. Для того чтобы запустить движение, нужно переключиться на точку наблюдения Rotation.

Интерполятор положения

PositionInterpolator(интерполятор положения) дает возможность ме-нять с течением времени положение объекта в пространстве. Он пересчитывает трехмерные значения и может быть маршрутизирован на поле set_translation узла Transform. Впро-чем, его можно маршрутизировать на любое поле типа SFVec3f.

Скалярный интерполятор

ScalarInterpolator (скалярный интерполятор) соединяет действительные значения. С его помощью можно влиять на интенсивность света, прозрачность объекта или любую другую скалярную величину.

Прототипирование

Прототипирование – это особый способ повторного использования кода. Если вы хотите получить ряд объектов одного типа, вы можете применить DEF и USE.

Если нужен ряд объектов с почти одинаковыми свойствами и очень небольшими различиями, на пример цвета, то можно использовать PROTO.

Как и в случае с другими XML – ориентированными языками, буква X в аббревиатуре X3D означает eXtensible (расширяемый). В отличие от большинства языков, которые имеют фиксированную лексику, X3D позволяет осуществить прямое расширение своего языка путем использования прототипов для построения, объявления и использования новых узлов.

Прототипы позволяют создавать новые типы узлов, которые могут использоваться многократно, как и любой другой узел. В определении прототипа, размечается новый узел с необходимыми полями. Он конструируется из других узлов включая экземпляры ранее созданных прототипов. Далее это определение используется, для создания экземпляров.

Это – мощная возможность, делающая X3D расширяемым языком разметки.

Объявленные прототипы можно размещать в отдельном X3D-файле или в библиотечном архиве. Это осуществляется при помощи узла ExternProtoDeclare.

Такой подход позволяет улучшать (дорабатывать) прототипы в отдельном файле, а в других файлах его многократно использовать. Управление такими библиотеками прототипов в больших проектах более практично и удобно в сопровождении.

Поскольку прототипы позволяют определять поля с изменяемыми значениями и вставлять код сценария, они являются очень полезными, когда необходимо иметь X3D-узлы с переменными параметрами (полями) и поведением. Часто используемые комбинации узлов и модели систем – являются хорошими кандидатами для прототипирования.

Можно создавать и постоянно усовершенствовать (без изменения интерфейса) прототипы чтобы они больше соответствовали целям проекта.

Каждый новый узел может быть настроен при инициализации (создании экземпляров) с использованием узлов <fieldValue> для переопределения значений полей по умолчанию.

В некотором отношении прототип обеспечивает повторные возможности для декларативного графа сцены аналогичным способом, которым подпрограмма, метод или функция обеспечивает возможности повторного использования, т.е. многократного вызова с новыми значениями входных параметров.

При создании новой копии узла используется узел <ProtoInstance> который создает новый тип X3D-узла. В его дочернем узле <fieldValue> задаются значения полей.

Узел внутренней <ProtoDeclare> или внешней <ExternProtoDeclare> протодекларации должен предшествовать любому соответствующему узлу <ProtoInstance> инициализации (создания) экземпляра, используемого в сцене.

Это требование позволяет браузеру распознавать каждый новый узел.

В процессе прототипирования необходимо строго контролировать типы узлов. Каждый заявленный узел прототипа соответствует типу первого (корневого) узла в теле прототипа.

Это позволяет строгого контролировать типы узлов во время загрузки (проверяя родительско-дочерние отношения узлов). Таким образом, новым экземплярам прототипа можно вписаться в граф сцены только в тех местах, где разрешен первый узел тела прототипа. Другие узлы могут следовать в теле объявления прототипа, но только первый узел определяет используемый тип. Кроме того, только первый узел визуализируется.

Требования строгого контроля типов предотвращают недопустимые (внутренне противоречивые) ошибки. Самый плохой вид ошибки – скрытая ошибка. Браузеры могут проверить, соответствуют ли экземпляры прототипа должным образом типу узла. Это – важный шаг в гарантии того, что графы сцены с прототипами созданы без ошибок.

Для повторного использования служат также Inline-узлы. Их можно использоваться вместо объявления прототипа и его реализации. Фактически, Inline-механизм – простой и легкий способ скопировать внешние подграфы сцены в сцену эталона.

Однако, внедренные Inline-узлы не могут быть изменены во время создания в локальном графе сцены, потому что у них нет механизма определения полей.

Таким образом, Inline-узлы менее сильны чем прототипы, и не расширяют X3D как язык. Однако, Inline-узлы стали более сильными в случае их использования с узлами IMPORT и EXPORT.

Эти узлы-интерфейсы позволяют использовать ROUTE в пределах родительской сцены с целью посылать события соответствующим полям в пределах содержащих Inline-узел (и также получать события вывода).

ProtoDeclare

Объявление прототипа определяет интерфейс и тело прототипа. Интерфейс определяет поля узла прототипа для хранения информации, её записи и считывания из экземпляра прототипа.

Эти поля описываются с помощью имени (name), типа значения (type), типа доступа (accessType) и значения по умолчанию (value).

Тело прототипа состоит из X3D-узлов, в качестве которых могут выступать экземпляры других (ранее определённых) прототипов.

Каждый прототип должен иметь уникальное имя. Хорошие имена прото-узлов и их полей очень важны. Давая прототипам уникальные имена, которые описывают их назначение, приводят к понятному назначению сцены. Хорошие имена позволяют другим лучше понимать новый прототип. Хорошие имена обеспечивают ясность и естественное мышление о том, что происходит в сцене. Хорошо подобранные имена прото-узлов и их полей способствуют эффективной реализации проекта и помогают разработчикам избегать ошибок, создавая экземпляры прототипа. Имена узла должны быть уникальными в каждой сцене X3D.

ProtoInterface

Узел ProtoInterface определяет все поля, которые составляют интерфейс прото-узла. Каждый дочерний узел field содержит имя нового поля, тип его значений, тип доступа и возможное начальное значение по умолчанию.

Интерфейс прототипа может не иметь полей или иметь несколько полей. По умолчанию прототип полей не имеет. Определения поля для объявлений прототипа идентичны определениям поля для узлов Скрипта.

Поля ProtoInterface отвечают следующим правилам, которые относятся к полям встроенным в узлы X3D:

- У initializeOnly - полей должно быть начальное значение;
- У inputOutput - полей должно быть начальное значение;
- У inputOnly - полей нет никакого начального значения;
- У outputOnly - полей нет никакого начального значения.

Эти правила также относятся к полям, определенным в пределах узлов Скрипта.

Отметим, что узлы ROUTE не могут соединиться непосредственно с определенными полями в интерфейсе прототипа. Они могут только подключить поля узлов, которые определены в пределах тела прототипа.

В узле <field> могут использоваться поля appinfo и documentation. Их значения позволяют обеспечивать дополнительные метаданные об определениях.

Поля appinfo и documentation определения следуют за шаблоном проекта аннотации, обеспеченным XML-схемой. Это часто используется в формировании документации при создании новых XML-элементов и атрибутов.

Поле appinfo используется для создания описания, которое является пользовательской подсказкой в создаваемых приложениях. Поле documentation используется для детального описания или url-ссылки на дополнительную информацию.

В следующем фрагменте X3D-кода приведены примеры использования описанных полей.

```
<ProtoDeclare>

<ProtoInterface>
  <field name='enabled' type='SFBool' value='true' accessType='inputOutput'
  appinfo='Whether or not ViewPositionOrientation sends output to console.'/>
  <field name='position_changed' type='SFVec3f' accessType='outputOnly'
  appinfo='Output local position.'/>
  <field name='orientation_changed' type='SFRotation' accessType='outputOnly'
  appinfo='Output local orientation.'/>
</ProtoInterface>

<ProtoBody>
    <!—ProtoBody definition goes here—>
</ProtoBody>

</ProtoDeclare>
```

Узел field требует наличия полей name, accessType и type. В зависимости от accessType может также быть необходимо задавать значение поля по умолчанию. Возможны четыре значения accessType: initializeOnly, inputOnly, outputOnly и inputOutput. Они соответствуют выборам, доступным для каждого узла и поля в X3D, включая определенные автором поля, описанные в секции узла Скрипта. Отметим, что любому новому определению поля с accessType="initializeOnly" или accessType="inputOutput" нужно определить начальное значение.

<ProtoBody>

Узлы, которые составляют функциональные возможности прототипа, входят в тело прототипа. Эти узлы формируют подграф сцены, который подключается в родительский граф сцены всякий раз, когда загружается экземпляр прототипа. Первый узел в <ProtoBody>, определяет тип узла прототипа. Этот тип узла позже используется для проверки правильности родительско-дочерних отношений в момент создания экземпляра прототипа. Дополнительные узлы могут следовать как часть тела объявления, но только первый узел имеет право на визуализацию. Это важное ограничение рендеринга гарантирует, что неподходящее поведение не следует из других несовместимых узлов, которые могут быть рядом в получающемся графе сцены.

Рассмотрим IS-подключения для полей узлов ProtoBody и ProtoInterface. Поля в теле прототипа связываются с полями в интерфейсе прототипа. Для этого применяют узел <IS> и его дочерний узел <connect>. IS-подключение, связывающее внутреннее поле тела с интерфейсом прототипов, автоматически связывает значения между ними. Отметим, что IS/connect -определения должны соответствовать и accessType и type.accessType определяет, инициализированы ли значения, или изменения их значений сопровождаются событиями. Значения initializeOnly и inputOutput поля accessType требуют, чтобы значения инициализировались. Значение initializeOnly не передают события. Значения inputOnly, outputOnly и inputOutput поля accessType могут передать события.

Вложенные объявления прототипа могут запутывать разметку. Поэтому лучше избегать вложенных протообъявлений. На практике лучше определить все прототипы на верхнем уровне сцены. Использование уже заявленных экземпляров прототипа в теле другого прототипа уменьшает возможные проблемы вложения прототипов и образования их имён.

У подграфа сцены прототипа есть свое собственное пространство имен, и таким образом на копии узлов USE нельзя сослаться между телом прототипа и родительским графом сцены. На практике авторы, используют уникальные названия DEF в родительской сцене и прототипе.

Перегруженные названия DEF в единственном файле вызывают ошибку проверки правильности XML, потому что два идентичных типа XML-идентификатора создают конфликт имен DEF.

Отметим, что IS-подключения, может использоваться для любого поля интерфейса, включая поля, имеющие accessType = "inputOnly" и accessType = "outputOnly".

Названия прототипов могут совпадать, если каждый определен и используется в отдельных файлах сцены. Это позволяет использовать Inline-узлы.

Одинаково названные прототипы могут также быть использованы через url в ExternProtoDeclare. Однако, перегрузка имен узла не допустима непосредственно в пределах единственной сцены.

Внешние объявления прототипа: ExternProtoDeclare

Внешние объявления прототипа позволяют многократно использовать одно определение прототипа из другого файла. Каждое определение ExternProtoDeclare почти идентично интерфейсу ProtoDeclare.

ExternProtoDeclare добавляет поле url для того, чтобы восстановить оригинальный прототип, возможно со многими адресами кандидата. Это также опускает определение значений по умолчанию для каждого поля, потому что они уже определены в ProtoDeclare (и может измениться, не предупреждая автора ExternProtoDeclare).

Этот подход не позволяет задавать недопустимые значения по умолчанию. Поле url может содержать множественные адреса. На конкретные прототипы можно сослаться по имени через использование символа (#), который сопровождает имя прототипа.

Экземпляры прототипа: <ProtoInstance>

Экземпляр прототипа создает новый узел в графе X3D-сцены во время выполнения, основанном на определении ProtoDeclare нового узла.

Название узла чувствительно к регистру и должно соответствовать своему определению. Во время загрузки создается и инициализируется параметризованная копия прототипа, которая использования как экземпляр узла в графе сцены.

Узлы ProtoInstance могут иметь DEF-имена и USE-копии как любой другой узел.

У экземпляров прототипа есть все возможности как и у регулярных встроенных X3D-узлов.

Синтаксис кодирования <ProtoInstance> имеет вид

```
<Group>
  <ProtoInstance name='ViewPositionOrientation‘ containerField = 'children'>
      <fieldValue name='enabled' value='true'/>
 </ProtoInstance>
</Group>
```

Узел fieldValue используется (если это необходимо) для отмены заданных по умолчанию значений и назначения новых значений в момент инициализации. Тип узла ProtoInstance установлен значением поля containerField.

Это значение containerField должно соответствовать соответствующему имени поля в родительской вершине ProtoInstance. В XML-кодировании – это поле со значением по умолчанию containerField = "children" (наиболее общая ценность), который может быть отменен как необходимый.

Рекомендуется прототипировать работающий (отлаженный и всесторонне протестированный X3D-код), т.е. следует преобразовывать существующий код в объявление прототипа.

Для этого выделенный фрагмент графа сцены может быть обернут в ProtoDeclare и ProtoBody со своим название прототипа.

На этом этапе поддающиеся изменению поля могут быть идентифицированы через определения поля в узле ProtoInterface.

Наконец, узлы IS подключают определения, и значения по умолчанию для полей, что завершает объявление прототипа.

Связанные узлы и понятия

Inline-узлы могут также использоваться для многократного использования геометрии и поведения. Inline-узлы не могут быть изменены или настроены при создании, хотя события можно послать с узлом ROUTE в или из Inline-узла через определения IMPORT и EXPORT.
Узел Скрипта часто используется для настройки поведения злов прототипа.

С помощью прототипов можно создавать новые типы узлов, в которых можно менять поля, так называемые "переменные". Структура определяющая новый прототип узла имеет вид:

```
<ProtoDeclare name='имя_прототипа'>
    <ProtoInterface> Список полей </ProtoInterface>
    <ProtoBody> Тело прототипа </ProtoBody>
</ProtoDeclare>
```

Список полей – это список параметров, которыми будет обладать новый узел. Поля описываются в следующем узле:

```
<field accessType='типДоступа' name='уникальноеИмя' type='типПоля' value='значениеПоУмолчанию'/>
```

Здесь использованы следующие поля:

accessType (тип доступа) – это поле может иметь следующие значения: initializeOnly, inputOnly, outputOnly или inputOutput, которые указывают предназначение и режим доступа к полю.

name (имя) – уникальное имя которое мы присваиваем полю <field>.
type (тип) – под типом значений подразумеваются следующие зарезервированные слова.

Тип поля	Значения по умолчанию
SFBool	False
MFBool	
SFColor	0 0 0
MFColor	
SFColorRGBA	0 0 0 0
MFColorRGBA	
SFInt32	0
MFInt32	
SFFloat	0.0
MFFloat	
SFDouble	0.0
MFDouble	
SFImage	0 0 0
MFImage	
SFNode	
MFNode	
SFRotation	0 0 1 0
MFRotation	
SFString	
MFString	
SFTime	-1
MFTime	
SFVec2f/SFVec2d	0 0
MFVec2f/MFVec2d	
SFVec3f/SFVec3d	0 0 0
MFVec3f/MFVec3d	

Примеры полей:

```
<!-- поле с именем NUMBER хранящее целое (не изменяемое) число. По умолчанию поле принимает значение 0 -->
<field accessType='initializeOnly' name='NUMBER' type='SFInt32' value='0'/>
<!-- изменяемые поля size и color -->
<field accessType='inputOutput' name='color' type='SFColor' value='0 0 0'/>
<field accessType='inputOutput' name='size' type='SFFloat' value='0.0'/>
<!-- изменяемые поля -->
<field accessType='inputOnly' name='MOVE' type='SFVec3f' value='0 0 0'/>
<field accessType='inputOnly' name='ROTATE' type='SFRotation' value='0 1 0 0'/>
<field accessType='outputOnly' name='ROTATING' type='SFBool' value='false'/>
```

В тело прототипа вставляется описание, создаваемого узла. К примеру можно задать построение сложной формы, параметры которой будут зависеть от конкретных значений полей этого узла.

Литература:

1. Введение в X3D. http://tellcomp.ru/book/4 - http://tellcomp.ru/book/11