S. A. Belyaev, A. V. Ekalo
Saint Petersburg Electrotechnical University «LETI»

E. A. Rubtsov, S. A. Kudryakov
Saint Petersburg State University of Civil Aviation

METHOD OF COVERAGE EVALUATION FOR CIVIL AVIATION RADIO SYSTEMS WHEN SELECTING THE POSITION OF THEIR PLACEMENT

Presents a method of coverage evaluation for civil aviation radio systems, allowing to take into account terrain relief and atmospheric parameters of a considered region. Using the digital elevation model SRTM allows to evaluate the closing angles for the selected position without having to departure to the place for the theodolite survey of the closing angles. The control example for radar 1L118 shows good convergence of simulation results and direct measurements. The parameters of the atmosphere are estimated from the data of long-term observations given in aero climatic reference books. Carrying out of calculations is possible on the average indicators or indicators of the worst month. Considering method can be used to select the optimal positions for radar stations, short-range navigation systems VOR/DME, ground-based automatic dependent surveillance stations, and VHF radio communication radio systems to support existing as well as future flight safety requirements.

Coverage, closing angle, digital elevation model, SRTM, positioning of radio systems, aviation safety

УДК 681.3.07

Т. В. Герасимова
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Веб и 3D-графика

Рассматривается возрастающая роль применения трехмерной графики как важной части мультимедийного веб-контента. Представлен и подробно рассмотрен обзор современного уровня техники использования 3D-графики в реальном времени в Интернете, охватывающий методы рендеринга, методы описания сцены и предоставления 3D-данных. Представлены подходы к 3D-рендерингу на основе браузера, классифицированные по уровню декларативного поведения. Показана эволюция стандарта X3D, позволяющего получить более низкий уровень доступа к графическому оборудованию, постоянно увеличивающему мощность. Рассмотренные удаленные методы рендеринга позволяют передавать высококачественную 3D-графику на широкий спектр устройств, а в последние годы также широко изучаются методы доставки контента для веб-3D-приложений. Все эти разработки отражаются в увеличении количества областей, для которых разрабатываются приложения для 3D-сети. Представлен широкий набор технологий, существующих в этой области.

3D, браузер, плагин, HTML5, WebGL, X3D, X3DOM, XML, Three.JS, Java, веб, рендеринг, canvas

Веб – интернет-пространство, прошедшее долгий путь с момента появления. Первоначально оно существовало как средство совместного использования удаленных страниц статического текста через Интернет, связанных вместе языком разметки – HTML. Выпуск браузера Mosaic позволил пользователям «просматривать» удаленные «веб-страницы», в которых была представлена смесь текста и изображения [1]. После выпуска Netscape Navigator и Microsoft Internet Explorer в середине девяностых развитие веб-технологий пошло ускоренными темпами. Каскадные таблицы стилей (CSS), Secure Sockets Layer (SSL), файлы cookie, Java и Javascript, Adobe Flash, XML и AJAX – это только некоторые из используемых технологий. К концу тысячелетия сеть была полна профессионально разработанных страниц, богатых визуальным контентом. Однако интерак-

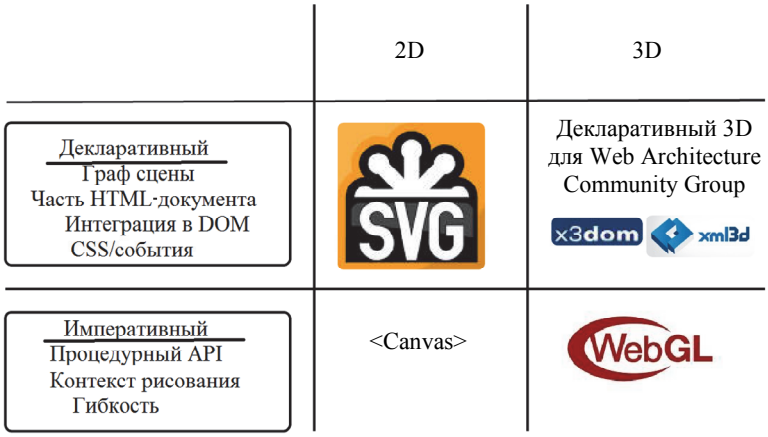

Рис. 1

тивный мультимедийный контент зависел от Adobe Flash [2]. Закрытая система Adobe позволила разработчикам встраивать интерактивные аудио-, видео- и 2D-анимации в исполняемый Flash-файл, который затем запускался плагином-браузером, установленным на пользовательском компьютере, почти полностью обойдя сам браузер. Приложения на основе Flash обычно требовали (относительно) длительного времени загрузки, и этот факт в сочетании с более поздним отсутствием их доступности для платформы iOS (Apple) означал, что Flash потеряла часть своей первоначальной популярности [3]. Между тем, в начале нового тысячелетия выросли возможности для создания интерактивных мультимедийных веб-страниц с использованием методов открытых стандартов. В браузеры была введена масштабируемая векторная графика (SVG), позволяющая использовать сложный 2D-рисунок, вписываемый в существующий стиль HTML.

Затем был добавлен элемент canvas (холст), который также позволяет выполнять 2D-рисунок, но отличающийся от SVG управлением через Javascript. Впервые представленный Apple как часть структуры WebKit canvas позже был включен в проект стандарта HTML5 [4] (наряду с SVG). HTML5 специально разработан для адаптации HTML таким образом, чтобы использоваться для создания веб-приложений, т. е. динамических интерактивных страниц, которые богаты мультимедийным контентом.

Термин в компьютерной графике «рендеринг» описывает процесс преобразования трехмерных данных в 2D-изображение на экране отображения. Методы визуализации сильно различаются с точки зрения их сложности, скорости, фотореализма и применения. Фотореализм обычно оценивается в соответствии с реализмом визуализации 3D-объекта, а также тем, как этот объект затенен по отношению к источникам света в сцене. Комплексные методы затенения, такие, как трассировка лучей и излучение, создают 2D-изображения, которые являются более фотореалистичными за счет увеличения времени вычисления. Уменьшая сложность алгоритмов, используемых для имитации эффекта света в сцене, время рендеринга может быть сокращено, чтобы позволить приложениям с 2D-изображениями визуализироваться достаточно быстро. В предлагаемой статье рассмотрим обзор современного уровня техники использования 3D-графики в реальном времени в Интернете, охватывающий методы рендеринга, методы описания сцены и предоставления 3D-данных.

Рендеринг на основе браузера. Существует классификация трехмерного рендеринга на основе браузера (т. е. где клиентский браузер/машина выполняет процесс рендеринга) в декларативные и императивные техники, создавая классификации 2D- и 3D-графики в соответствии с различными парадигмами (рис. 1) [5].

Например, можно рисовать 2D-графику в браузере с использованием SVG и элемента canvas HTML5. SVG представляет собой формат файла на основе XML2D-векторной графики – изображения и эффекты фильтров изображения закодированы декларативным образом. Напротив, подобный процесс рисования и обработки изображений может быть достигнут с помощью элемента <canvas> императивным способом с использованием Javascript.

В таблице представлены подходы к 3D-рендерингу на основе браузера, классифицированные по уровню декларативного поведения. Настраиваемый конвейер – это тот, где программист имеет низкоуровневый контроль над объектами рендеринга, порядок визуализации, граф сцены, ма-

Подход	Требуется плагин	Часть HTML-документа	Интеграция DOM	Встроенный граф сцены	Настраиваемый конвейер*	Базирующиеся на стандартах
X3D	Да	Нет	Нет	Да	Нет	Да
X3DOM	Нет	Да	Да	Да	Нет	Да
XML3D	Нет	Да	Да	Да	Нет	Частично
O3D	Больше нет	Нет	Нет	Да	Нет	Частично**
Three.JS	Нет	Нет	Нет	Да	Да	Частично**
Stage3D (Flash)	Да	Нет	Нет	Нет	Да	Нет
Java	Да	Нет	Нет	Да	Да	Нет
Silverlight	Да	Нет	Нет	Нет	Да	Нет
Native WebGL	Нет	Нет	Нет	Нет	Да	Да
Unity3D	Да	Нет	Нет	Да	Нет	Нет

териалы и связанные шейдеры(*). O3D и Three.JS могут отображаться через WebGL, который основан на стандартах(**). Unity3D включен в качестве (популярного) примера патентованных игровых движков.

VRML, X3D и стандарты ISO. В 1994 г. был разработан формат файла виртуальной реальности (VRML) для описания 3D-сцены, а формат (со второй версии) стал стандартом ISO (ИСО/МЭК 14772-1: 1997) в 1997 г. [6]. VRML97 использует текст для указания как содержимого, так и внешнего вида сцены (например, связывания сетки с конкретным материалом), но не позволяет специфицировать такие более сложные 3D-понятия, как NURBS (неявные поверхности) или сложная анимация. Вскоре после определения VRML97 и для того, чтобы защитить его как открытый стандарт, был сформирован консорциум Web3D как совокупность предприятий, академических институтов и частных лиц. Было разработано несколько приложений и плагинов, чтобы отображать VRML-сцены в таких браузерах, как Cosmo Player, World-View, VRMLView и Blaxxun Contact. Во многих отношениях эти приложения сформировали первые приложения для введения 3D-графики в Интернет. В 2004 г. VRML был заменен на X3D (ISO/ IEC19775/19776/19777). Несмотря на то, что X3D поддерживает обратную совместимость с VRML, он предоставляет более продвинутые API, дополнительные форматы кодирования данных, более строгие соответствия и компонентную архитектуру [7]. Наибольшим отличием от VRML является синтаксис: в то время как X3D по-прежнему поддерживает традиционный синтаксис VRML C-like, он также добавляет поддержку двоичных и XML-форматов. Для просмотра и взаимодействия со сценариями X3D требуется приложение с поддержкой X3D (автономное программное обеспечение или плагин браузера). Интеграция с веб-браузером включает в себя браузер, содержащий сцену внутри, позволяя разработчику плагина контролировать и обновлять контент через интерфейс доступа к экрану.

X3DOM. В попытке расширить поддержку браузера для X3D в 2009 г. был [8] представлен X3DOM. X3DOM предоставляет собственный браузер без плагинов (где это возможно) и широкие возможности 3D. Он предназначен для тесной интеграции со стандартными веб-технологиями, такими, как AJAX, поддерживает декларативный характер X3D, пытается скопировать X3D непосредственно на веб-страницу без необходимости в плагине браузера. X3DOM определяется как фронт- и бэкенд-система, где компонент соединителя преобразует сцену, определенную в интерфейсе (DOM), в бэкенд (X3D). Затем соединитель берет на себя ответственность за синхронизацию любых изменений между двумя окончаниями модели. X3DOM определяет пространство имен XML [9] включением тега <x3d> (и всех его дочерних элементов), который будет использоваться на стандартной странице HTML или XHTML. X3DOM предназначен для интеграции с DOM и поэтому может быть изменен таким же образом, как и любой объект DOM [10]. 3D-модели могут быть загружены в X3D/X3DOM, определяя геометрию вершин или импортируя сцену X3D (или VRML), определенную в отдельном файле.

Ниже приведен пример кода, необходимого для создания сцены с образцом сетки, начиная с исходного тега <body> файла XHTML (для краткости выводятся данные вершин объекта).

```
<body>
  <X3D

xmlns="http://www.web3d.org/specifications/x3d-
namespace"
    width="400px" height="400px">
  <Scene DEF='scene'>
      <Viewpoint position='0 0 300'
            orientation="0 0 0 1" />
      <Background skyColor='0.6 0.6 0.6'/>
```

```
        <Transform translation='0 10 0' >
        <Shape>
          <Appearance DEF='App_0'>
              <Material diffuseColor="0 1 0" shiniess='0.15625'/>
          </Appearance>
          <IndexedFaceSet creaseAngle='4' coordIndex='
              [indexed vertex coordinates for model]
              '/>
         </IndexedFaceSet>
        </Shape>
    </Transform>
  </Scene>
</X3D>
    <script type="text/javascript" src="x3dom.js">
</script>
</body>
```

Рис. 2

Результат этого кода при загрузке в веб-браузере показан на рис. 2. Этот фрагмент кода использует иерархическое объявление для создания 3D <scene>. Позиция и ориентация точки обзора заданы, как и цвет фона. Тег <transform> определяет преобразования трансляции, вращения и масштабирования, которые применяются к содержимому тега (в этом случае весь контент, определенный внутри него, должен быть перенесен на 10 единиц по оси *y*). Позиция и ориентация точки обзора заданы, как и цвет фона. Тег <shape> определяет объект в сцене, в котором свойства объекта (такие, как сетка и внешний вид) определяются параллельно. Сетка в этой сцене определяется как набор индексированных граней; никакие нормальные векторы не поставляются, поэтому X3DOM вычисляет их для включения освещения. Для рендеринга сцены X3DOM определяет резервную модель, пытаясь сначала инициализировать предпочтительные визуализаторы и при необходимости использовать альтернативную технологию. Модель резервной проверки сначала проверяет, может ли пользовательское приложение отображать X3D-код изначально, и в этом случае передает содержимое тега <x3d> в механизм рендеринга X3D, определенный хост-приложением.

Если встроенный рендеринг X3D не обнаружен или плагин для браузера X3D не установлен, приложение пытается создать контекст рендеринга WebGL, а также построит граф сцены X3D поверх этого. Большинство современных браузеров теперь поддерживают 3D-рендеринг на основе браузера через WebGL.

X3DOM предназначен для интеграции с DOM и поэтому может быть изменен таким же образом, как и любой объект DOM. Например, если программист хотел изменить положение фигуры во время выполнения (например, в ответ на ввод пользователя), атрибут перевода тега <transform> может быть изменен во время выполнения с использованием Javascript, и сцена будет обновляться соответствующим образом. 3D-модели могут быть загружены в X3D/X3DOM, определяя геометрию вершины, как в приведенном ранее примере, или импортируя сцену X3D (или VRML), определенную в отдельном файле. Этот файл сцены может быть закодирован вручную, а для сложных объектов и сцен он может быть создан с использованием пользовательских модулей экспорта для создания 3D-контента (DCC) (таких, как Blender, Autodesk Maya и Autodesk 3D Studio Max).

Основное затенение (например, диффузное или зеркальное) поддерживается декларативно. Пользовательские шейдеры поддерживаются через узел ComposedShader; это позволяет программисту написать свой собственный код шейдера. Позднее были введены несколько методов навигации камеры, которые определяют узел точки обзора и несколько различных шаблонов навигации (определение пользовательской навигации камеры также поддерживается посредством манипуляции эквивалентным узлом DOM, например с помощью Javascript). Также была представлена простая анимация объектов, либо с помощью CSS-преобразований и анимацией, либо с помощью интерполяторов X3D.

XML3D. XML3D [11] похож на X3DOM, поскольку он является расширением HTML, предназначенным для поддержки интерактивной 3D-графики в браузере, требующей использования XHTML. Его цель – попытаться найти минимальный набор дополнений, которые полностью поддерживают интерактивный 3D-контент как неотъемлемую часть 2D/3D-веб-документов. Можно выделить следующие характеристики XML3D:

– является расширением для HTML5;

– повторно использует элементы HTML5, например элемент <img> для определения данных текстуры. Содержимое XML3D может быть оформ-

лено с помощью CSS2, т. е. XML3D поддерживает все соответствующие свойства стиля;

– для сценариев определяет интерфейсы DOM. Все элементы XML3D происходят из интерфейса DOM HTMLElement;

– использует UIEvents как свою систему событий и поддерживает все релевантные события, доступные в HTML. Кроме того, он поддерживает все соответствующие атрибуты событий;

– использует элемент <defs> из SVG.

XML3D использует аналогичный подход высокого уровня к 3D-программированию, как и X3DOM. Центральное различие между этими двумя подходами состоит в том, что X3DOM возникла из-за попытки встроить существующую инфраструктуру (X3D) в контекст браузера, тогда как XML3D предлагает, по возможности, расширить существующие свойства HTML, встраивая 3D-контент в веб-страницу с помощью максимального повторного использования существующих функций.

Пример кода XML3D ниже рисует 3D-сцену (для краткости исключается стандартный html-код, такой, как теги тела и головы, а также строки для загрузки библиотек XML3D):

```
</head>
<body
        <div id="overall">
<div id="content">
        <!--<h1>Spot Light - urn:xml3d:lightshader:spot</h1>-->
            <xml3d id="MyXml3d" class="xml3d" view="#defaultView">
              <defs>
                 <transform id="t_spotLight1" rotation="0.0 1.0 0.0 0.37963513" translation=
                                          "–9.05869 –1.965812 6.827833"></transform>
                <transform id="t_directionalLight1" rotation="0.0 1.0 0.0 0.62446165">
                                                                        </transform>
             </defs>
             <group id="VisualSceneNode">
                 <mesh material="lamp.xml#Kabel-fx" src="lamp.xml#Line09Shape" type=
                                                  "triangles"> </mesh>
                <mesh material="basic.xml#BodenMaterial-fx" src="basic.xml#BodenShape"
                                                   type="triangles"></mesh>
                <group id="spotLight1" transform="#t_spotLight1">
                    <view id="lightView">
                          <float name="fovVertical">1.2</float>
                     </view>
                     <light model="urn:xml3d:light:spot">
                              <float3 name="intensity">1.0 1.0 1.0</float3>
                              <float3 name="attenuation">1.0 0.0 0.015</float3>
                              <float name="cutoffAngle">0.8</float>
                              <float name="softness">10</float>
                              <bool name="castShadow">true</bool>
                              <float name="shadowBias">0.01</float>
                         </light>
                </group>
                <light model="urn:xml3d:light:directional" transform="#t_directionalLight1">
                          <float3 name="intensity">0.1 0.1 0.1</float3>
                </light>
                <mesh material="lamp.xml#Metall-fx" src=
                                    "lamp.xml#LampShape" type="triangles"> </mesh>
                <mesh type="triangles" style=
                "transform: translate3d(-14px, 0px, 1px)" material="lamp.xml#Metall-fx">
                          <data src="basic.xml#mesh_cube"></data>
                </mesh>
                <transform id="cameraTransform" translation=
                          "-26 -31 22" rotation="0.8242830634117126 -0.3122488260269165
                           -0.47229036688804626 1.0554853677749634"></transform>
                <view id="defaultView" transform="#cameraTransform"></view>
         </group>
</xml3d>.
```

В этом примере тег <defs> позволяет определять различные преобразования, данные для сетки, а также свет (положение и интенсивность) и положение зрения. Преобразования определяются как тег <transform id>, как и в X3DOM, хотя они могут быть указаны с использованием преобразований CSS для тех браузеров, которые их поддерживают, выполняя цель – как можно больше использовать существующие функции. Результат этого кода при загрузке в веб-браузер показан на рис. 3.

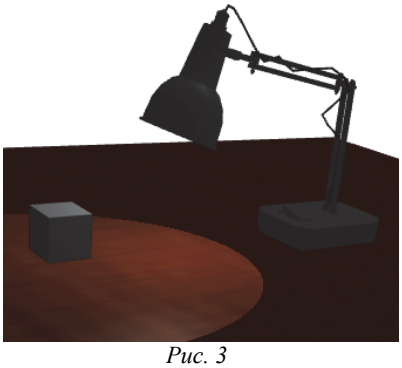

Рис. 3

Шейдер можно указать с помощью элемента <shader>. Как и элемент <mesh>, <shader> может иметь произвольные параметры. Он также ссылается на узел скрипта через атрибут script. Этот узел сценария содержит используемый шейдерный код. XML3D поставляется с набором стандартных шейдеров стандартной функции, которые имеют предопределенный набор параметров. Эти шейдеры ссылаются на URN. Текущие реализации поддерживают только эти фиксированные шейдеры функции. В более поздней версии разрешено использование шейдерных скриптов с помощью системы AnySL. Шейдер можно назначить группе, используя свойство shader. Затем все потомки будут отображаться с использованием этого шейдера, если только свойство шейдера не будет перезаписано дочерними группами. Вот пример того, как Phong shader задан и назначен группе:

```
<xml3d id="MyXml3d" style="width: 150px; height: 100px; border: 1px solid gray"
 xmlns="http://www.xml3d.org/xml3d">
  <defs>
    <shader id="ex4_1_redShader" script="urn:xml3d:shader:phong">
      <float3 name="diffuseColor">1.0 0.5 0.5</float3>
      <float name="ambientIntensity">1.0</float>
    </shader>
  </defs>
    <group style="shader: url(#ex_redShader)">
      <mesh type="triangles">
      <int name="index">0 1 2  1 2 3</int>
      <float3 name="position">-1 -1 -5 1 -1 -5 -1 1 -5 1 1 -5</float3>
      <float2 name="texcoord">0.0 0.0 1.0 0.0 0.0 1.0 1.0 1.0</float2>
      </mesh>
    </group>
</xml3d>.
```

XML3D также вводит другой уровень абстракции с помощью Data Containers, определяя тег <data>, который обертывает примитивные данные. Это определение контейнеров декларативных данных позволяет обрабатывать модули обработки данных, что в свою очередь допускает сложные преобразования геометрии, необходимые для динамических сеток и некоторых графических эффектов.

XML3D-рендеринг реализован как в WebGL, так и в качестве встроенного компонента для браузера Mozilla (поддерживающего Firefox) и браузеров на основе браузера Webkit (поддерживаемых Google Chrome и Apple Safari). XML3D использует и интегрируется с другими спецификациями и стандартами W3C[12].

CSS 3D-Transforms. CSS (Cascading Style Sheets), или каскадные таблицы стилей, используются для описания внешнего вида документа, написанного языком разметки. Обычно CSS-стили служат для создания и изменения стиля элементов веб-страниц и пользовательских интерфейсов, написанных на языках HTML и XHTML, но также могут быть применены к любому виду XML-документа, в том числе XML, SVG и XUL. Каскадные таблицы стилей описывают правила форматирования элементов с помощью свойств и допустимых значений этих свойств. Для каждого элемента можно использовать ограниченный набор свойств, остальные свойства не будут оказывать на него никакого влияния.

Преобразования CSS являются частью спецификации W3C и позволяют преобразовывать элементы, созданные с помощью CSS-кода, в двух- или трехмерном пространстве. Первоначально разработанные Apple как часть среды WebKit [12] (функционирующие в браузерах Safari и Chrome), эти преобразования теперь полностью поддерживаются браузерами Mozilla и почти полностью поддерживаются в Internet Explorer.

3D-преобразования работают, сначала устанавливая перспективу в сцене (с использованием transform: perspective(value); или более просто perspective: value ;). Свойство активирует 3D-пространство для элемента. Свойство `perspective` и функция `perspective()` добавляют глубину элементу, увеличивая его размеры

по оси *z*, сам элемент при этом становится визуально меньше. Чем меньше значение, тем ближе *Z*-пространство к зрителю и тем больше эффект, заданный с помощью свойства `transform`.

Если 3D-перспектива задается с помощью функции `perspective()`, 3D-пространство активизируется только для одного элемента. Свойство `perspective` активирует 3D-пространство внутри элемента, содержащего дочерние трансформированные элементы, и применяется к ним. Свойство не наследуется.

Функция преобразования perspective важна для трехмерных преобразований. Она задает расположение смотрящего и размещает просматриваемое содержимое на пирамиде просмотра, которая в итоге проецируется на двухмерную плоскость. Если перспектива не задана, все точки в пространстве *Z* переносятся на ту же двухмерную плоскость, и в итоге преобразование не создает ощущения объемности. Для некоторых преобразований, таких, как перенос вдоль оси *z*, функция преобразования перспективы очень важна для различения любых эффектов преобразований.

После установки стандартные 3D-преобразования, такие, как перенос, вращение и масштабирование, могут применяться в трех осях. Параметр *z* в основном заставляет элемент двигаться ближе и дальше, в зависимости от уменьшения или увеличения значения. Это как увеличение и уменьшение масштаба:

```
@keyframes zooming {
  0%   { transform: translate3d(0, 0, 0); }
  100% { transform: translate3d(0, 0, 200px); }
}
p { animation: zooming 5s alternate; }
```

На рис. 4 показана работа программы – прозрачный зеленый блок поднимается на 200px «*вверх*» по оси *z*, как будто становясь ближе к нам.

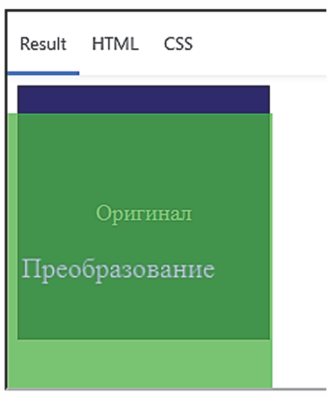

Рис. 4

Используя CSS-переходы, может быть достигнута простая анимация между различными состояниями. Затенение объектов должно указываться вручную, определяя информацию о цвете или текстуре (как изображение), как со стандартным CSS.

В приведенном ниже примере блок изменит цвет фона в течение одной секунды в линейном виде:

```
.box {
  background: #2db34a;
  transition-property: background;
  transition-duration: 1s;
  transition-timing-function: linear;
}
.box:hover {
  background: #ff7b29;
}
```

CSS 3D-преобразования обеспечивают очень быстрый и понятный способ кодирования простых 3D-эффектов на веб-страницу. Одним из интересных аспектов их использования является то, что существующий 3D-контент, такой, как элемент DOM (например, холст), который имеет 3D-графику, представленную другими методами, может быть далее преобразован с использованием CSS 3D.

Тем не менее отсутствие истинных возможностей освещения и затенения означает, что CSS 3D очень ограничен в том, что может быть достигнуто с помощью более мощных декларативных решений, таких, как X3DOM или XML3D.

Javascript-доступ к графическому оборудованию. В отличие от декларативного или функционального программирования парадигма императивного программирования описывает вычисления как серию утверждений, которые изменяют состояние программы. Традиционным языком для выполнения недекларативного кода в веб-браузере является Javascript, который содержит элементы императивных, объектно-ориентированных и функциональных парадигм программирования. Одним из самых важных факторов, способствующих возникновению императивного 3D-программирования в Интернете, является огромное увеличение производительности в Javascript Virtual Machine, позволяющее быстро контролировать и манипулировать тысячами вершин в трехмерном пространстве, в каждом рисованном кадре [13]. Хотя можно создать алгоритм рендеринга веб-программного обеспечения с использованием SVG [14] или холста HTML5, к концу первого десятилетия XXI в. делались попытки разрешить программный доступ к выделенному

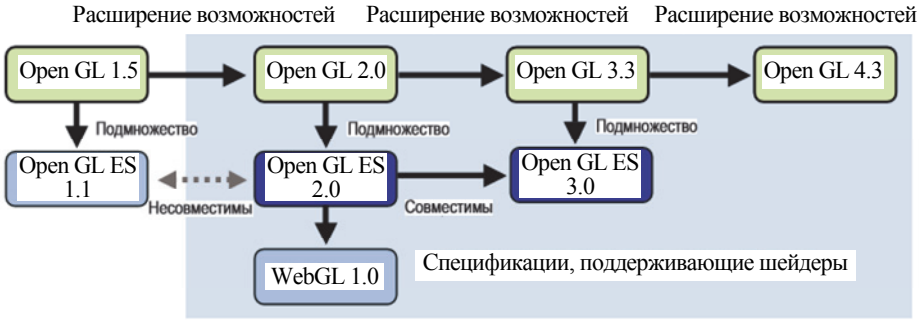

Рис. 5

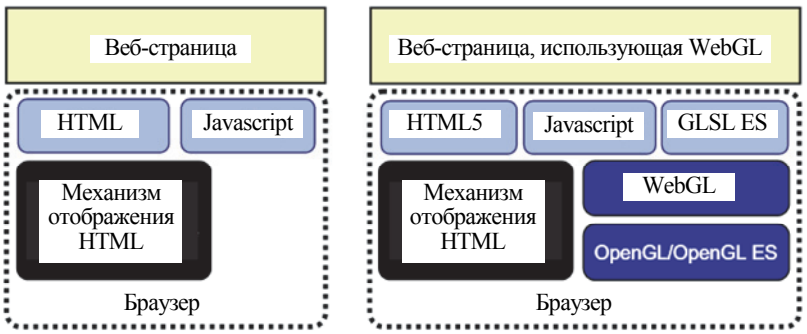

Рис. 6

графическому оборудованию. Среди них была библиотека O3D Google [15] [16]. Разработанный как межплатформенный плагин для всех основных браузеров и операционных систем, O3D первоначально предоставил Javascript API для доступа к его коду плагина (написанному на C/C ++), который, в свою очередь, позволял программировать графическое оборудование либо через Direct3D, либо через OpenGL (решение было скрыто от конечного пользователя). Все перечисленные выше разработки используют какой-либо вид плагина браузера, обычно запрограммированный на C или C ++, который по существу действует как обертка Javascript для доступа к графическому оборудованию через OpenGL или Direct3D. К 2009 г. стала очевидной необходимость стандартизации методов доступа к графическому процессору через браузер.

WebGL (Web-based Graphics Library). Группа разработчиков Khronos описывает WebGL таким образом: **WebGL** – это межплатформенный бесплатный веб-стандарт для низкоуровневого 3D-графического API на основе OpenGL ES 2.0, который отображается через элемент HTML5 Canvas в качестве интерфейсов Document Object Model [17]. OpenGL ES («Встроенные системы») 2.0 – это адаптация стандартного API OpenGL, разработанного для устройств с ограниченной вычислительной мощностью, таких, как мобильные телефоны или планшеты.

Переход к версии OpenGL 2.0 означился появлением новой важной особенности – поддержкой программных шейдеров. Эта поддержка была перенесена в OpenGL ES 2.0 и стала одним из основных элементов спецификации WebGL 1.0. На рис. 5 показана связь между OpenGL, OpenGL ES 1.1/2.0/3.0 и WebGL.

Язык программирования, используемый для создания шейдеров, называется языком шейдеров (shading language). Язык шейдеров, определяемый в спецификации OpenGL ES 2.0, основан на языке шейдеров OpenGL (GLSL) и называется языком шейдеров OpenGL ES (GLSL ES). Так как WebGL основана на OpenGL ES 2.0, в ней, для создания шейдеров, также используется язык GLSL ES.

WebGL следует клиент-ориентированному подходу рендеринга для рендеринга 3D-сцен. Вся обработка, требуемая для получения изображения, выполняется локально с помощью графического оборудования клиента. WebGL предназначен для использования в сочетании со стандартными веб-технологиями; таким образом, пока 3D-компонент веб-страницы рисуется с помощью API WebGL через Javascript, сама страница сформирована стандартным HTML. На рис. 6 представлена структура традиционной динамической веб-страницы (слева) и веб-страницы, использующей WebGL.

С появлением WebGL возникла необходимость добавить в эту комбинацию язык шейдеров GLSL ES. Это означает, что веб-страницы, использующие технологию WebGL, создаются на трех языках: HTML5, JavaScript и GLSL ES.

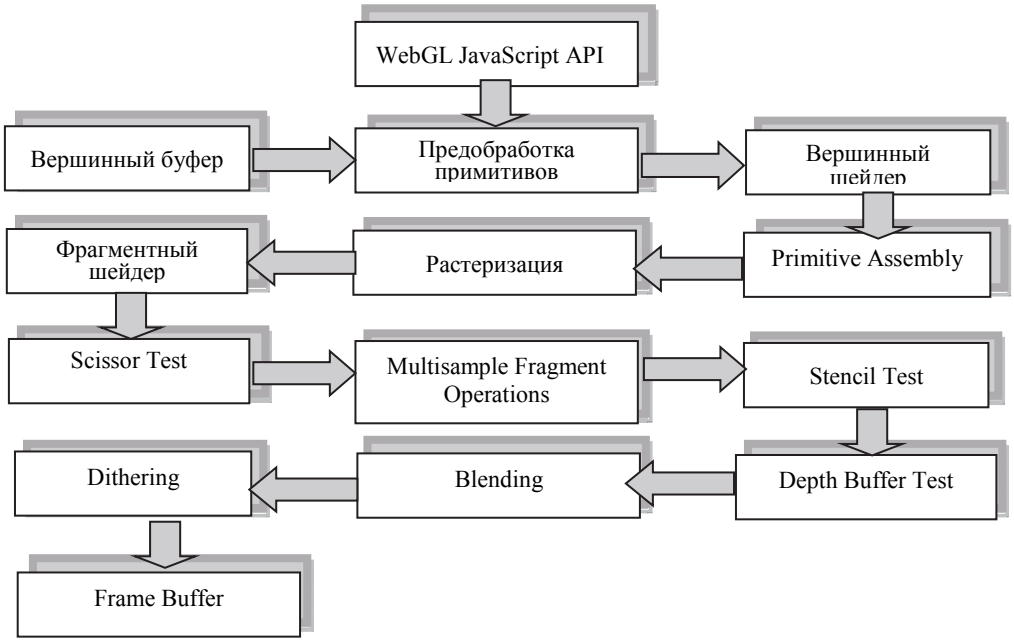

Рис. 7

Графический конвейер WebGL. Конвейер WebGL схематично можно представить следующим образом (рис. 7):

• Вначале создаем набор вершин в буфере вершин (Vertex Buffer). По этим вершинам впоследствии будут составлены геометрические примитивы, а из примитивов – объекты. Далее проводим некоторую предобработку.

• Затем содержимое буфера вершин поступает на обработку в вершинный шейдер (Vertex Shader). Шейдер производит над вершинами некоторые трансформации, например применяет матрицы преобразования и т. д.

• На следующем этапе (Primitive Assembly) конвейер получает результат вершинного шейдера и пытается измененные вершины сопоставить в отдельные примитивы – линии, треугольники, спрайты. Также на этом этапе определяется, входит ли примитив в видимое пространство.

• Далее на этапе растеризации (Rasterization) полученные примитивы преобразуются в фрагменты, которые можно представить как пиксели, в дальнейшем отрисованные на экране.

• Затем в дело вступает фрагментный шейдер (Fragment shader). Он производит преобразования с цветовой составляющей примитивов, наполняет их цветом, точнее окрашивает пиксели, и в качестве вывода передает на следующий этап измененные фрагменты.

• Scissor Test – на этом этапе проверяется, находится ли фрагмент в пределах отсекающего прямоугольника.

• Multisample Fragment Operations – на данном этапе у каждого фрагмента изменяются цветовые составляющие, производится сглаживание (anti-alising).

• Stencil Test – здесь фрагмент передается в буфер трафаретов (stencil buffer). В этом буфере дополнительно отбрасываются те фрагменты, которые не должны отображаться на экране.

• Depth Buffer Test – тест буфера глубины.

• Blending – на данном этапе происходит небольшое смешение цветов, например для создания прозрачных объектов.

• Dithering – здесь происходит смешение цветов для создания тонов и полутонов.

• Frame Buffer – здесь наконец полученные после предобработки фрагменты превращаются в пиксели на экране.

WebGL запускается на графическом процессоре на компьютере пользователя, который предоставляет код, работающий на этом графическом процессоре, в виде пар функций. Эти две функции называются вершинным шейдером и шейдером фрагментов, и каждый из них написан на очень строго типизированном языке C/C++, называемом GLSL (GL Shader Language).

Рассмотрим вопрос применения шейдеров в WebGL для реализации освещенности в 3D. Первое, что нужно сделать, это обновить вершинный шейдер, чтобы он генерировал значение затенения для каждой вершины графического объекта (куб) на основе окружающего освещения, а также направленного освещения. Посмотрим на код (фрагмент программы):

```
const vsSource = `
    attribute vec4 aVertexPosition;    attribute vec3 aVertexNormal;    attribute vec2 aTextureCoord;
    uniform mat4 uNormalMatrix;    uniform mat4 uModelViewMatrix;    uniform mat4 uProjectionMatrix;
    varying highp vec2 vTextureCoord;    varying highp vec3 vLighting;
    void main(void) {
        gl_Position = uProjectionMatrix * uModelViewMatrix * aVertexPosition;
        vTextureCoord = aTextureCoord;
      // Apply lighting effect
        highp vec3 ambientLight = vec3(0.3, 0.3, 0.3);
        highp vec3 directionalLightColor = vec3(1, 1, 1);
        highp vec3 directionalVector = normalize(vec3(0.85, 0.8, 0.75));
    highp vec4 transformedNormal = uNormalMatrix * vec4(aVertexNormal, 1.0);
    highp float directional = max(dot(transformedNormal.xyz, directionalVector), 0.0);
    vLighting = ambientLight + (directionalLightColor * directional);
};
```

Как только вычисляется положение вершины, передаем координаты текселя, соответствующие вершине, в шейдер фрагмента и можем работать над вычислением затенения для вершины.

Первое, что делаем, – преобразовываем нормаль, основанную на текущей ориентации куба, умножая нормальную вершину на нормальную матрицу. Затем можно вычислить количество направленного освещения, которое должно быть применено к вершине, вычислив точечное произведение преобразованной нормали и вектора направления (т. е. направления, с которого идет свет). Если это значение меньше нуля, привязываем его к нулю. В результате получим значение RGB, которое будет использоваться фрагментным шейдером для настройки цвета каждого пикселя, который визуализируем.

Теперь необходимо обновить шейдер фрагмента, чтобы принять во внимание значение освещенности, вычисленное вершинным шейдером:

```
const fsSource = `
    varying highp vec2 vTextureCoord;    varying highp vec3 vLighting;
    uniform sampler2D uSampler;
    void main(void) {
        highp vec4 texelColor = texture2D(uSampler, vTextureCoord);
        gl_FragColor = vec4(texelColor.rgb * vLighting, texelColor.a); };
```

Перед установкой цвета фрагмента умножаем цвет текселя на значение освещенности, чтобы настроить цвет текселя и учесть эффект источника света пользователя. Результат программы, включающей приведенный код реализации шейдеров при загрузке в веб-браузер, показан на рис. 8.

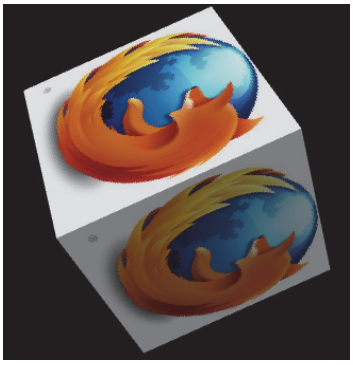

Рис. 8

Three.JS. Возможно, самая известная библиотека/API для веб-графики 3D.3JS [18], [19]. Хотя она была первоначально разработана в ActionScrip, теперь представляет собой библиотеку Javascript с открытым исходным кодом, которая позволяет осуществлять высокоуровневое программирование трехмерных сцен на основе браузера. Ее модульная структура означает, что несколько различных изображений (WebGL, Canvas и SVG) были разработаны для рендеринга содержания сцены, и библиотека может расширяться распределенным образом несколькими десятками отдельных участников. Three.JS содержит граф сцены, несколько типов камер и режимов навигации, несколько предварительно запрограммированных шейдеров и материалов (а также возможность программирования пользовательских шейдеров), загрузку и рендеринг LOD-сетки, а также компонент анимации, позволяющий выполнять скелетную и морфинг-анимации.

Стоит объяснить, почему часто в настоящее время выбирается three.JS, а не чистый WebGL. Причин несколько. Первая и главная – быстрый результат. Формирование сцены с визуализацией 3D-моделей осуществляется значительно быстрее, чем в WebGL. Three.js же выполняет всю низкоуровневую работу с WebGL за пользователя, предоставляя набор классов с понятным интерфейсом для удобной работы с 3D-графикой, а кроме того, она позволяет быстро достичь результата, не разбираясь во всех сложностях и тонкостях в отображении материалов в WebGL. Скачать библиотеку и посмотреть множество интересных примеров можно на ее официальном сайте (http://www.threejs.org). На рис. 9 показан пример включения в сцену освещения, которое получено с использованием основных типов источников света в three.JS.

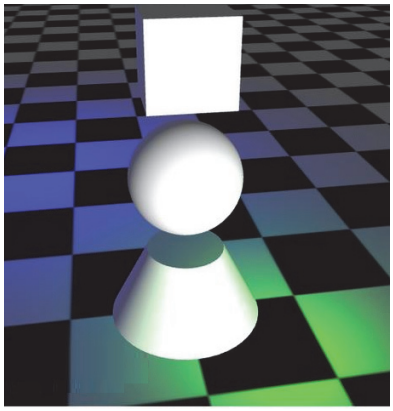

Рис. 9

Сцена содержит 4 источника света (по одному каждого типа) и отображает простую черно-белую текстурированную плоскость основания и 3 простых белых геометрических объекта. На сцене также можно увидеть эффекты от освещения точечным и прожекторным типами источников света. Наконец, окружающий свет обеспечивает небольшое количество освещения для всех объектов сцены одинаково. Далее показан фрагмент кода настройки освещения:

```
// Create and add all the lights
        directionalLight.position.set(.5, 0, 3);
        root.add(directionalLight);
        pointLight = new THREE.PointLight
(0x0000ff, 1, 20);
        pointLight.position.set(−5, 2, −10);
        root.add(pointLight);
        spotLight = new THREE.SpotLight
(0x00ff00);
        spotLight.position.set(2, 2, 5);
        spotLight.target.position.set(2, 0, 4);
        root.add(spotLight);
        ambientLight = new THREE.AmbientLight (
0x888888 );
        root.add(ambientLight);
```

Как и почти все остальное в WebGL, источники света являются искусственно созданной конструкцией. WebGL знает только о буферах и шейдерах, разработчикам необходимо синтезировать световые эффекты, написав шейдерный код. Three.JS предлагает потрясающий набор материалов и возможностей освещения, он был написан на Javascript. Конечно, все это было бы невозможно, если бы WebGL не предоставил пользователю доступ к графическому процессору, чтобы в первую очередь создать эти удивительные эффекты.

При использовании WebGL существует несколько вариантов того, как именно визуализировать изображения. Например, API рисования 2D Canvas позволяет использовать *Z*-буферизацию рендеринга там, где аппаратное обеспечение использует дополнительную память для рисования только тех пикселей, которые расположены впереди в сцене или нет. Это выбор разработчика. Если не использовать *Z*-буферизацию, приложению придется самостоятельно сортировать объекты, потенциально вплоть до уровня треугольника. Это звучит как большая проблема, но в зависимости от варианта использования можно сделать именно так. Это лишь один из выборов, который можно сделать в отношении рендеринга.

Stage 3D. Как уже упоминалось, Adobe Flash-плагин представляет собой запатентованную систему, которая позволяет мультимедийному контенту работать внутри веб-страницы с включенным плагином Flash. Несмотря на первоначальные попытки встраивать 3D-графики во Flash (например, исчезнувший Papervision [7]), разработчики полагались на методы рендеринга программного обеспечения, которые не позволяли получить доступ к графическому процессору. Stage 3D – собственный 3D-движок Adobe [8], причем ключевое отличие заключается в том, что он позволяет приложениям Flash и AIR рисовать аппаратную ускоренную 3D-графику. Приложения AStage 3D написаны в ActionScript, объектно-ориентированном языке, разработанном для написания приложений на основе Flash.

Silverlight. Silverlight предоставляет графическую систему, схожую с Windows Presentation Foundation, и объединяет мультимедиа, графику, анимацию и интерактивность в одной программной платформе [20].

Он был разработан, чтобы работать с XAML и с языками Microsoft .NET. XAML применяется для разметки страниц, использующих векторную графику и анимацию. Текст, содержащийся в приложениях Silverlight, доступен для поисковых систем, так как он не компилируется, а доступен в виде XAML. Silverlight также можно использовать для того, чтобы создавать виджеты для Windows Sidebar в Windows Vista. Silverlight позволяет динамически загружать XML и использовать DOM для взаимодействия с ним. Silverlight содержит объект Downloader, благодаря которому можно скачивать скрипты, медиа-файлы и т. д., если это необходимо приложению. Начиная с

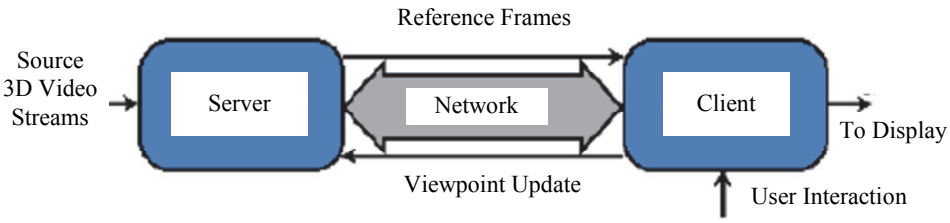

Рис. 10

версии 2.0 логика программы может быть описана в любом из языков .NET, включая динамические языки программирования, такие, как Iron Ruby и Iron Python, которые в свою очередь исполняются в DLR (Dynamic Library Runtime). Разработка Silverlight велась последовательным добавлением функциональности в каждой новой версии. Так, Silverlight 4 представлял собой надмножество над Silverlight 3, а та в свою очередь являлась надмножеством над Silverlight 2. Код для данной платформы не обладает полной совместимостью между версиями, главным образом из-за вынужденного применения программистами различных обходных путей при отсутствии какой-либо встроенной функциональности. Причем добавление такой функциональности в последующей версии весьма негативно может повлиять на корректную работу примененной ранее «хитрости», из-за чего такой код, как правило, приходится переписывать.

В некоторых довольно редких случаях интерфейс какой-либо функциональности может измениться из-за реализации нового, более удачного решения. Тем не менее подобные резкие изменения довольно редки и перенос кода на новую версию платформы происходил сравнительно безболезненно.

Java. Платформа Java, разработанная Sun Microsystems до ее слияния с Oracle, теперь является частью современной вычислительной техники. С точки зрения Интернета, возможность запуска Java-апплета (небольшого приложения, которое выполняется в виртуальной машине Java) в браузере была одним из самых ранних способов программирования более дорогостоящих визуальных вычислений [9]. В частности, можно разрешить Java-апплетам получать доступ к графическому процессору, который до появления WebGL был одним из ранних способов доступа к аппаратным ускорениям с веб-страницы, не полагаясь на собственный плагин браузера. Java3D API был выпущен для облегчения разработки 3D-приложений с помощью Java [21]. Java3D имеет полный граф сцены и способен визуализировать с помощью Direct3D или OpenGL. Разработка на Java3D была прекращена в 2008 г., так как Sun Microsystems переместилась на новую платформу JavaFX, хотя разработка на Java3D была перезапущена сообществом JogAmp [22].

Удаленный рендеринг. Концепция удаленного рендеринга включает в себя создание данных на основе сервера, которые затем передаются клиенту для визуализации (рис. 10).

Большая часть исследований в этой области не относится непосредственно к веб-3D-графике, поскольку она фокусируется на системах рендеринга «клиент-сервер», где клиент обычно не является веб-приложением.

Однако все чаще веб-браузер используется в качестве клиента 3D-графики, поэтому методы удаленного рендеринга представляют интерес. При классификации различных подходов к удаленному рендерингу можно выделить 3 области: графические команды, пиксели и примитивы или векторы.

Графические команды. Низкоуровневые вызовы рисования на сервер GPU перехватываются и передаются клиенту [23], который затем осуществляет рендеринг и отображает окончательное изображение. Этот метод был адаптирован [24] для параллельной визуализации в WebGL для существующих 3D-настольных приложений с использованием AJAX при синхронизации двух процессов рендеринга. Кроме того, существует прямое преобразование кода C/C ++ (через байт-код LLVM) в Javascript [25].

Пиксели. Сервер осуществляет рендериг изображения и передает его непосредственно клиенту для отображения [26]–[28]. Этот базовый метод удаленного рендеринга можно рассматривать как общую проблему передачи данных, по существу, выборку буфера удаленной системы и отправку его клиенту в виде видеопотока [29]. В этой области можно сделать несколько оптимизаций, таких, как синхронизация высокопроизво-

дительного рендеринга сервера с низкоуровневым рендерингом клиента [30], выборочная передача пикселей [31] или оптимизация кодирования видео для использования графических сцен с графическим процессором [32].

Примитивы или векторы. Методы выделения объектов используются на сервере для получения векторов, которые должны быть переданы клиенту для рендеринга, либо в 2D [33], либо в 3D [34]. Преимущество этого метода заключается в том, что клиентские устройства, которые не имеют собственных 3D-возможностей, могут отображать 3D-объекты из переданных векторных данных, поскольку дорогостоящие 3D-преобразования выполняются сервером.

Можно утверждать, что мир веб-графики 3D является ярким и захватывающим как для исследователей, разработчиков, так и для пользователей. Каждый последующий год приносит дальнейшие исследования и разработки, которые распространяются онлайн, в общей академической прессе и конференциях, а также в конкретных мероприятиях, таких, как Международная конференция по 3D Web Technology, которая в 2017 г. прошла уже в 22-й раз.

СПИСОК ЛИТЕРАТУРЫ

1. Schatz B. R., Hardin J. B. NCSA Mosaic and the World Wide Web // Global Hypermedia Protocols for the Internet. Science. 1994. № 265 (5174). P. 895–901.
2. Curtis H. Flash Web Design: The Art of Motion Graphics. New Riders Publishing, 2000. 256 p.
3. Nielsen J. Designing Web Usability. New Riders Publishing, 1999. 419 p.
4. W3C. HTML5.2 Specification. URL: http://www.w3.org/TR/html5/ (дата обращения 19.02.2018).
5. Declarative Integration of Interactive 3D Graphics into the World-WideWeb: Principles, Current Approaches, and Research Agenda / J. Jankowski, S. Ressler, Y. Jung, J. Behr, P. Slusallek. URL: https://hal.inria.fr/hal-00822667/document (дата обращения 08.05.2018).
6. Geroimenko V., Chen C. Visualizing Information Using SVG and X3D. Springer, 2004. URL: http://bookfi.net/book/1297537/ (дата обращения 01.06.2018).
7. Web3D. X3D. 2017. URL: http://www.web3d.org/sites/default/files/page/About%20Web3D%20Consortium/What_Is_X3D_July2017/ (дата обращения 01.03.2018).
8. X3dom-web3d2009-paper.pdf..http://www.web3d.org/wiki/images/3/30/ (дата обращения 19.02.2018).
9. Adobe. Stage 3D. 2011. URL: http://www.adobe.com/devnet/flashplayer/stage3d.html (дата обращения 19.02.2018).
10. Namespaces in XML 1.0 (Third Edition) W3C Recommendation 8 Dec. 2009. URL: https://www.w3.org/TR/xml-names/ (дата обращения 10.05.2018).
11. XML3D_interactive_3D_graphics_for_the_web. URL: https://www.researchgate.net/publication/ (дата обращения 10.05.2018).
12. XML3D. Specification. 2017. URL: http://xml3d.org/xml3d/specification/latest/ (дата обращения 01.03.2018).
13. Parisi T. WebGL: Up and Running. URL: https://the-eye.eu/public/Books/IT%20Various/ (дата обращения 10.05.2018).
14. Tautenhahn L. SVG-VML-3D. 2002. URL: http://www.lutanho.net/svgvml3d/ (дата обращения 01.03.2018).
15. Google. O3D. 2008. URL: https://code.google.com/p/o3d/ (дата обращения 10.05.2018).
16. Ortiz Jr. S. Is 3D Finally Ready for the Web? // Computer. 2010. № 43 (1). С. 14–16.
17. Khronos. WebGL Specification. 2017. URL: http://www.khronos.org/registry/webgl/specs/latest/1.0/ (дата обращения 01.03.2018).
18. Cabello R. Three.JS. 2010. URL: https://threejs.org/docs/index.html#manual/introduction/ (дата обращения 05.03.2018).
19. Dirksen J. Learning Three.js: The JavaScript 3D Library for WebGL. Packt Publishing, 2013. 453 p.
20. Silverlight 5. URL: https://www.microsoft.com/silverlight/ (дата обращения 05.03.2018).
21. Dalton Filho. 3D Model Interaction with Java 3D. URL: https://dzone.com/articles/3d-model-interaction-java-3d/ (дата обращения 05.03.2018).
22. JogAmp. JogAmp. 2013. URL: http://jogamp.org/. (дата обращения 19.02.2018).
23. WireGL: a scalable graphics system for clusters / G. Humphreys, M. Eldridge, I. Buck, G. Stoll // Proc. of the 28th annual conf. on Computer graphics and interactive techniques (SIGRAPH), New York, 2001. P. 129–140.
24. ReWeb3D-Enabling Desktop 3D Applications to Runinthe Web. URL: http://cadcamcae.eafit.edu.co/documents/ (дата обращения 15.02.2018).
25. Emscripten, 2014. URL: http://emscripten.org/. (дата обращения 15.02.2018).
26. An accelerated remote graphics architecture for PDAS. URL: http://sanna.polito.it/Versioni_Postscript/ (дата обращения 15.03.2018).
27. Streaming Scenes to MPEG-4 Video-Enabled Devices. URL: http://www.cs.tau.ac.il/~dcor/online_papers/papers/ (дата обращения 15.03.2018).
28. MPEG-4-based adaptive remote rendering for video games / N. Tizon, C. Moreno, M. Cernea, M. Preda // Proc. of the 16th Intern. Conf. on 3D Web Technology – Web3D'11, ACM New York, USA, 2011. P. 45.
29. Virtual network computing / T. Richardson, Q. Sta_ord-Fraser, K. R. Wood, A. Hopper // Internet Computing. IEEE. 1998. № 2 (1). P. 33–38.
30. Levoy M. Polygon-assisted JPEG and MPEG compression of synthetic images. URL: https://graphics.stanford.edu/papers/poly// (дата обращения 15.03.2018).

31. Mann Y., CohenOr D. Selective pixel transmission for navigating in remote virtual environments. URL: https://www.cs.tau.ac.il/~dcor/articles/older/ (дата обращения 01.03.2018).

32. Fechteler P., Eisert P. Depth map enhanced macroblock partitioning for H. 264 video coding of computer graphics content. URL: http://iphome.hhi.de/eisert/papers/ (дата обращения 01.03.2018).

33. Moser M., Weiskopf D. Interactive Volume Rendering on Mobile Devices. URL: https://pdfs.semanticscholar.org/f917/ (дата обращения 01.03.2018).

34. Using expressive rendering for remote visualization of large city models / J. C. Quillet, G. Thomas, X. Granier, P. Guitton, J. E. Marvie // URL: https://hal.inria.fr/inria-00106832v2/document/ (дата обращения 21.02.2018).

T. V. Gerasimova
Saint Petersburg Electrotechnical University «LETI»

WEB AND 3D GRAPHICS

The increasing role of 3D graphics as an important part of multimedia web content is considered. In the proposed work, a review of the current state of the art of using real-time 3D graphics in the Internet, covering rendering methods, methods for describing a scene and providing 3D data, is presented and discussed in detail. Approaches to 3D-rendering based on the browser, classified by the level of declarative behavior are presented. The evolution of the X3D standard is shown, which allows to obtain a lower level of access to graphic equipment, which constantly increases the power. Considered remote rendering methods allow you to transfer high-quality 3D graphics to a wide range of devices, and in recent years also widely studied methods of delivering content for web-based 3D applications. All these developments are reflected in the increase in the number of areas for which applications for the 3D network are being developed.

3D, browser, plugin, HTML5, WebGL, X3D, X3DOM, XML, Three.JS, Java, web, rendering, canvas

УДК 681.3

А. М. Голубков
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Бинарная классификация изображений на примере задачи распознавания лиц

Рассматривается практическое применение теоремы о разделимости, утверждающей, что в многоразмерных пространствах произвольно выбранная точка этого пространства отделима линейным дискриминантом от любого произвольно выбранного множества точек этого пространства при достаточно большой размерности. Показано, что «достаточно большой» является размерность больше двадцати. В качестве приложения данной теоремы приводится задача распознавания (бинарной классификации) лиц людей. Используя сверточные нейронные сети, для каждой фотографии лица формируется вектор признаков (в данном случае 128-мерный). Каждый такой вектор можно рассматривать как точку в 128-мерном пространстве. Показано, что, используя теорему о разделимости, можно с вероятностью, близкой к единице, отделить лицо одного человека от лиц остальных людей. В качестве дискриминанта используется линейный дискриминант Фишера, который предоставляет одношаговую процедуру отделения, не требующую предварительного обучения. В результате экспериментов на наборе из 13 233 фотографий была получения точность 0.9980 с полнотой 0,9623.

Нейронные сети, бинарная классифкация, теорема о разделимости, линейный дискриминант, расстояние Махаланобиса

Бинарная классификация изображений является популярной задачей со множеством технических приложений. Решение данной задачи отвечает на вопрос, относится ли данное изображение к заданному классу или нет. Например, для актуальной задачи автопилотирования автомобилей