создать впечатляющую 3D графику в CSS3

Эта статья показывает, как использовать преобразования для создания и анимирования 3D объектов, используя только CSS.

CSS 3D преобразования быстро становятся признанным способом добавления глубины и интерактивности в веб-проекты.

В этом уроке мы поговорим о том, как можно подойти к процессу создания 3D сцены, построить 3D объект, добавить тень и анимацию, и внести динамический контент, чтобы создать что-то такое, что вы смогли бы использовать на вашем веб-сайте.

Также мы изучим несколько продвинутых возможностей CSS3 и рассмотрим некоторые общие вопросы.

Зачем использовать CSS?

Есть огромное количество способов создать 3D графику в HTML. На основе JS, Canvas, SVG, доступны даже решения на основе WebGL, и все они имеют свои достоинства, но есть еще и CSS.

Решение на базе CSS обходится без изображений и способно использовать графический процессор для ускорения обработки графики.

При этом не требуются плагины, и CSS может жить вне стен canvas на вашей странице:

Установка свойства perspective может произвести впечатляющий эффект

CSS 3D графика не так оптимизирована как WebGL. Отдельной темой является несовместимые браузеры. Тем не менее, включение 3D объектов в веб-проекты открывает новые направления работы, и может быть использовано уже сегодня.

Не все браузеры полностью поддерживают 3D преобразования, которые использованы в этой статье, поэтому для краткости я буду использовать префикс -webkit.

Не забудьте включить префиксы для Mozilla, Opera, MS, а также версии без префиксов в случае необходимости. Полные версии кода CSS (с префиксами) находятся в файлах проекта.

Основы: создание 3D куба

Для начала мы построим базовый 3D куб, разместив и анимировав его в 3D пространстве. Сначала напишем код HTML, который будет представлять части 3D куба:

<div class="stage">

<div class="cube">

<figure class="back"></figure>

<figure class="top"></figure>

<figure class="bottom"></figure>

<figure class="left"></figure>

<figure class="right"></figure>

<figure class="front"></figure>

</div>

</div>

Я задействовал элемент figure, но можно было использовать div, span или любой другой тип элемента на ваше усмотрение.

В этом HTML коде можно выделить две главные части: сцена, на которой все размещается, и куб, помещенный на сцену:

Слева – элемент div без применения преобразования. Справа – фигура повернута в 3D пространстве

Подготовка сцены

Первая вещь, которую необходимо сделать при построении 3D модели, это создать элемент, выступающий в качестве сцены.

Задаем глубину сцены с помощью CSS свойства perspective:

.stage {

width: 300px; height: 300px;

-webkit-perspective: 1600px;

-webkit-perspective-origin: 50% -240px;

}

Задание большей величины для свойства perspective создает менее заметный 3D эффект, меньшей – делает его более выраженным.

Значение в 800 пикселей или около того подходит, как правило, для большинства маленьких объектов, но если вы создаете что-то большее (например, небоскреб) вы можете использовать меньшее значение для создания более выраженных углов.

При задании свойства perspective необходимо также установить perspective-origin (свойство, определяющее положение виртуальной камеры, обращенной на сцену), позиционируемое на осях X и Y.

Создание форм: пространственное мышление

Создание 3D форм при помощи HTML и CSS включает в себя представление объекта в виде набора двухмерных плоскостей.

Вместо того чтобы рисовать вершины и собирать формы из точек и линий, что делается, как вам известно, в других 3D программах, мы будем работать с двухмерными плоскостями, позиционируя их одновременно. Мы собираем куб из шести плоскостей, соответствующих боковым сторонам, вершине и основанию.

Для начала придадим элементам figure форму и стиль:

figure {

display: block; position: absolute;

width: 300px; height: 300px;

background-color: #60c2ef;

}

На следующем шаге нужно указать, где они должны располагаться в 3D пространстве.

Время преобразований

Размещаем фигуры при помощи свойства transform. Преобразования в CSS состоят из последовательности инструкций, указывающих форме, как перемещаться, поворачиваться, наклоняться и изменяться в масштабе. Инструкции читают слева направо.

Например:

-webkit-transform: rotateY(45deg) translateZ(-100px) rotateX(10deg)

Мы можем поместить каждую сторону куба на место, начиная с передней и задней грани:

.cube { -webkit-transform-style: preserve-3d; }

figure { -webkit-transform-origin: 50% 50% 0; }

.front { -webkit-transform: translateZ(150px); }

.back { -webkit-transform: rotateY(180deg) translateZ(150px); }

Обратите внимание на использование свойства transform-style: когда оно установлено в значении preserve-3d, преобразования над сторонами куба будут осуществляться в их собственном 3D пространстве.

Мы также определяем точку, относительно которой выполняются преобразования.

Свойство transform-origin задает точку в X, Y, Z координатах, которая выполняет роль центра вращения (это середина передней части элемента):

Плоские элементы HTML в 3D пространстве

Передняя фигура смещена на 150 пикселей вперед, задняя – перевернута на 180 градусов и сдвинута на 150 пикселей к задней части сцены.

Порядок здесь имеет значение – правила преобразования применяются слева направо. Для второго преобразования задано положительное значение: после поворота передняя сторона обращена к задней части сцены.

Как и ранее порядок правил преобразований имеет значение. Если элемент повернут, его Z-ось будет находиться под другим углом, что повлияет на любое другое преобразование.

Теперь разместим другие стороны:

.top { -webkit-transform: rotateX(90deg) translateZ(150px); }

.bottom { -webkit-transform: rotateX(-90deg) translateZ(150px); }

.left { -webkit-transform: rotateY(-90deg) translateZ(150px); }

.right { -webkit-transform: rotateY(90deg) translateZ(150px); }

Теперь у нас полностью есть куб, расположенный на 3D сцене.

Двигай, двигай

Давайте повернем наше 3D творение на сцене с помощью CSS анимаций:

@-webkit-keyframes rotate {

0% { -webkit-transform: rotateY(0); }

100% { -webkit-transform: rotateY(360deg); }

}

CSS анимации создаются с помощью последовательности ключевых кадров. Ключевые кадры – это набор состояний объекта, установленных с использованием процентных отношений.

Анимация начинается с нулевого поворота и завершается полным поворотом элемента вокруг оси Y.

Примените это набор ключевых кадров со свойством animation:

.cube { -webkit-animation: rotate 10s infinite linear; }

Анимация поворота, примененная здесь, рассчитана на 10 секунд, повторяется бесконечно с постоянной линейной скоростью:

Готовый куб, созданный из шести HTML элементов

Затенение

У нас есть куб, вращающийся на сцене, но сейчас он освещен со всех сторон. Чтобы добавить тень, не задействовав больше селекторов, мы воспользуемся псевдо-селекторами.

Псевдо-селекторы – это что-то вроде дополнительных HTML элементов, которые можно вставить до и после содержимого элемента.

Они предоставляют нам дополнительные возможности и могут содержать градиенты, цвета или даже текст. Таким образом, мы можем анимировать затемнение на каждой стороне куба.

Нам нужно два градиента: один – для случая, когда поверхность поворачивается налево, другой – для поворота направо:

figure:before, figure:after {

content: ""; display: block; position: absolute;

width: 100%; height: 100%; opacity: 0;

}

figure:before {

background: -webkit-radial-gradient(top left, rgba(0, 0, 0, 0.5), #000000);

}

figure:after {

background: -webkit-radial-gradient(top right, rgba(0, 0, 0, 0.5), #000000);

}

Элементы before и after в этом примере являются псевдо-селекторами, которые могут быть стилизованы и анимированы.

Важно отметить, что эта функция еще не в полной мере поддерживается браузерами – Google Chrome, Firefox и IE10 предлагают различные уровни поддержки – и, возможно, ситуация улучшится.

Применяем анимацию к каждому псевдо-селектору для определения времени отображения/скрытия этих градиентов:

.front:before { -webkit-animation: frontShading1 10s infinite linear; }

...

Каждая анимация состоит из набора ключевых кадров, которые отображают или скрывают градиент во время анимации.

Ниже приведены ключевые кадры, примененные к задней стороне:

@-webkit-keyframes frontShading1 {

2% { opacity: 0; }

25% { opacity: 0.8; }

35% { opacity: 0.8; }

50% { opacity: 0; }

}

...

Полный набор анимаций, вместе с префиксами для каждого браузера, находится в файлах проекта.

Вершина и основание

Следующий шаг – вершина и основание куба. Для вершины нужен градиент, который выглядит так, как будто ничего не движется.

Добавьте градиент к элементу top и поверните его в противоположном направлении:

.top:after {

opacity: 0.5; width: 200%; height: 200%;

margin-left: -50%; margin-top: -50%;

background: -webkit-radial-gradient(bottom, rgba(0, 0, 0, 0), #000000);

-webkit-animation: rotateTop 10s infinite linear;

}

@-webkit-keyframes rotateTop {

0% { -webkit-transform: rotateZ(0); }

100% { -webkit-transform: rotateZ(360deg); }

}

Основание куба не будет видно, но мы можем использовать его, чтобы добавить атрибут box-shadow:

.bottom {

-webkit-box-shadow: 0 10px 100px rgba(0, 0, 0, 0.7);

background-color: rgba(0, 0, 0, 0);

}

Затенение добавляет чувство глубины в противоположность плоской сцене

Больше, чем просто цвет

К этому моменту мы, используя только CSS, создали затененный анимированный куб, правда, с однотонными сторонами. Поскольку мы работаем с HTML элементами, мы можем поиграть с содержимым.

Грани куба могут содержать изображения, текст или даже другие анимации. В примере, приведенном ниже, я применил несколько фоновых изображений из игры Minecraft:

figure {

background-image: url(../images/mc\_side.png);

background-size: 100%;

}

.top {

background-image: url(../images/mc\_top.png);

}

Затенение, которое мы добавили раннее, все еще присутствует здесь, придавая сторонам куба более сильное ощущение глубины. Этот эффект будет работать для любого контента элемента figure.

Отражения

Если анимированного затенения и тени недостаточно, мы можем добавить сцене немного больше глянца с помощью причудливых WebKit фильтров (учтите, что в настоящий момент эта функция возможна только с WebKit):

figure {

-webkit-box-reflect: below 0 -webkit-gradient(linear, left top, left bottom, from(transparent), color-stop(50%, transparent), to(RGBA(0, 0, 0, 0.1)));

}

Куб с твитами

Давайте создадим что-то более практичное. В этом примере мы сделаем куб местом отображения ваших самых последних твитов и поместим некоторую вторичную информацию на сторону куба с возможностью следовать за вами в Twitter.

Настройка HTML

Взяв за основу тот же код HTML, что и ранее, мы применим несколько фоновых изображений для придания кубу фактуры.

Используем следующие две текстуры с кубом:

figure {

background-color: #eeeeee;

background: url(../images/white-noise.jpg);

}

.front {

background: #e0e0e0 url(../images/net-logo.jpg) no-repeat;

Добавляем содержимое

Добавим содержимое, которое попадает в поле зрения пользователя при повороте куба. Это будет происходить на правой стороне куба. Подключаем в работу немного JavaScript для вызова кнопки «Follow» (доступна на странице кнопок Twitter).

Элемент figure с классом right должен содержать что-то вроде этого:

<figure class="right">

<p>The world's best selling magazine for web designers and developers since 1994</p>

<a href="https://twitter.com/netmag" class="twitter-follow-button" datashow-count="false">Follow @netmag</a>

<script>!function(d,s,id){var js,fjs=d.getElementsByTagName(s)[0],p=/^http:/.test(d.location)?'http':'https';if(!d.getElementById(id)){js=d.createElement(s);js.id=id;js.src=p+'://platform.twitter.com/widgets.js';fjs.parentNode.insertBefore(js,fjs);}}(document, 'script', 'twitter-wjs');</script>

</figure>

Кнопка «Follow» размещается путем добавления правила iframe в CSS:

.right iframe { padding-left: 90px; margin-top: -30px; }

Добавление фоновых изображений к HTML элементам придает сцене более презентабельный вид

Размещаем твит на кубе

Twitter предоставляет простой способ получения самых последних твитов в виде виджетов. Для установки виджета, войдите в учетную запись на Twitter и перейдите на страницу Twitter виджетов.

Выберите «Создать новую кнопку» (Create new button), затем введите нужный логин Twitter в поле «Имя пользователя» (Username). В этом примере мы используем логин netmag.

Нажмите «Создать кнопку» (Create widget button), чтобы сгенерировать встраиваемый код.

Мы можем стилизовать виджет и задать ширину, высоту и некоторые данные-значения в коде. Замените YOUR\_WIDGET\_ID на идентификатор, предоставленный в вашем коде для вставки:

<figure class="front">

<p style="padding: 0; margin: 85px 25px 0 25px; width: 250px">

<a class="twitter-timeline" href="https://twitter.com/netmag" width="250" height="200" data-chrome="noheader noborders nofooter transparent" data-tweet-limit="1" data-widget-id="YOUR\_WIDGET\_ID"></a>

<script>!function(d,s,id){var js,fjs=d.getElementsByTagName(s) [0],p=/^http:/.test(d.location)?'http':'https';if(!d.getElementById(id)){js=d.createElement(s);js.id=id;js.src=p+"://platform.twitter.com/widgets.js";fjs.parentNode.insertBefore(js,fjs);}}(document,"script","twitter-wjs");</script>

</p>

</figure>

Трансформации

Для затенения нам не потребуется анимация с помощью ключевых кадров: оно будет проявляться при наведении или возникновении некоторого события.

Добавим несколько переходов к затенению псевдо-селекторов:

.right:before { opacity: 0.4; -webkit-transition: all 0.4s ease; }

.right:after { display: none; }

.front:before { opacity: 0.1; -webkit-transition: all 0.4s ease; }

.front:after { display: none; }

Для псевдо-селектора на правой грани установлено значение непрозрачности 0,4. Это добавляет небольшую тень, и можно анимировать непрозрачность так, чтобы грань становилась светлее при повороте к лицевой стороне.

Псевдо-селектор right:after скрыт, так как мы не будем поворачивать куб на столько, чтобы он потребовался. Лицевой стороне придано затемнение, тем не менее, она светлее правой стороны, когда та поворачивается вперед:

Поворот 3D куба вокруг оси Y

Анимация при наведении

Теперь, добавив эффекты затенения, мы создадим анимированный эффект «подглядывания» когда пользователь наводит курсор на куб при помощи CSS состояния hover:

.cube:hover {

-webkit-transform: rotateY(-50deg);

-webkit-transition: all 0.4s ease;

}

.cube:hover figure.right:before, .cube:hover figure.front:before {

opacity: 0.2;

-webkit-transition: all 0.4s ease;

}

При наведении курсора куб поворачивается на 50 градусов влево. Это преобразование производится в виде плавного перехода путем применения свойства transition.

Одновременно с этим, непрозрачность псевдо-селекторов на передней и правой сторонах куба переходит в значение 0.2.

Результатом всех этих переходов является красивый эффект 3D поворота, позволяющий скрыть дополнительный контент – и показать его, если пользователь наведет курсор на куб.

Полный код HTML и CSS можно найти в файлах проекта:

Трехмерный «твит-куб», показывающий дополнительную информацию при наведении курсора

Создание объектов из плоских HTML элементов имеет ряд ограничений: любой объект, который вы создаете, необходимо сделать путем комбинирования плоских форм, чтобы получить конечный результат.

Искривленные объекты, такие как трубки и сферы, сложно создать без использования большого количества элементов, поэтому данное решение лучше подходит к более простым объектам.

Но прибегая к использованию теней и анимации, можно получить впечатляющие результаты и с простыми структурами.

Производительность и тестирование

При создании 3D объектов нужно помнить о том, что производительность браузеров и скорость компьютера могут очень сильно варьироваться. CSS, на самом деле, не оптимизировано для графически нагруженных, сложных 3D сцен, поэтому нужно быть осмотрительным при работе с публичными проектами.

Проводите тестирование на разных платформах: посетители могут также использовать мобильные устройства или устаревшие, медленные компьютеры.

И хотя это только вопрос времени, когда все браузеры будут поддерживать 3D преобразования, градиенты и все изящные штрихи, которые мы использовали здесь, реальность такова, что большинство браузеров все еще их не поддерживают.

Мы должны гарантировать, что посетители, пользующиеся старыми версиями браузеров, не получат искаженную страницу.

Одним из подходов для решения этой проблемы является использование обнаружения функций и отображение альтернативной версии для браузеров, которые не поддерживают нужные свойства.

В данном случае мы воспользуемся скриптом Modernizr для того, чтобы выяснить поддерживаются ли 3D преобразования, и если нет, отобразим статичное изображение.

Для начала подключите Modernizr в заголовке HTML:

<script src="js/modernizr.js"></script>

При загрузке в тег <head> будет вставлен класс csstransforms3d. Затем мы воспользуемся этой информацией для отображения или скрытия резервного варианта:

.stage { display: none; }

.csstransforms3d .stage { display: block; }

.fallback-stage {

width: 100%; height: 600px;

background: url(../images/fallback.png);

}

.csstransforms3d .fallback-stage { display: none; }

В этом примере альтернативой служит статичное изображение. Используйте запасной элемент div, чтобы представить двухмерный контент в браузере или альтернативное сообщение:

Текстуры, примененные к сторонам 3D куба с твитами

Следующие шаги

Стилизованный 3D куб – это, вероятно, не первое, что большинство заказчиков веб-дизайна просят сделать, но некоторые из приведенных подходов могут иметь другое применение.

Можно создать эффект 3D перехода при наведении на логотип – броский способ продемонстрировать новый продукт – или использовать данные подходы для представления и навигации в комплексных данных. Использование Z оси в дизайне открывает двери для многих интересных и веселых идей.

Перевод статьи «How to create impressive 3D graphics in CSS3» был подготовлен дружной командой проекта Сайтостроение от А до Я

Перейти к содержанию

htmlhook.ru

Скрипты для веб-приложений

Добавить рекламу

≡

Найти:

HTML

CSS

PHP

JS

jQuery

CMS

SEO

Шаблоны

Дизайн

Программы

Блог › JavaScript

▾ ›

Объекты 3D

В демонстрации показан мост между двухмерной и трехмерной графикой в HTML5 с помощью WebGL. Цель статьи показать, как нарисовать трехмерные объекты с помощью polygonal mesh. Polygonal mesh представляет собой набор вершин, ребер и граней, которые определяет форму многогранных объектов в 3D компьютерной графики и твердотельного моделирования.

▶Оглавление:

Озеро Самарцы - Отдых в Крыму для всей семьи!

Адрес: Россия, Республика Крым, Белогорский район, с. Зуя

http://samartsy.ru/

Простые трехмерные объекты

Читайте также: CSSrefresh

Фасад состоит из треугольников, четырехугольников или других простых многоугольников. Чтобы продемонстрировать это, мы подготовили простые трехмерные объекты — куб и многогранные сферы (с переменным числом граней).

Демо

HTML

Как обычно (для всех canvas на основе демо) у нас есть очень простая разметка HTML (с одним объектом внутри, canvas):

<html lang="ru" >

<head>

<meta charset="utf-8" />

<title>Каркас 3D объектов в HTML5</title>

<!-- стили -->

<link href="css/main.css" rel="stylesheet" type="text/css" />

<!-- скрипты -->

<script src="js/meshes.js"></script>

<script src="js/transform.js"></script>

<script>

//var obj = new cube();

//var obj = new sphere(6);

var obj = new sphere(16);

</script>

<script src="js/main.js"></script>

</head>

<body>

<div class="container">

<canvas id="scene" height="500" width="700" tabindex="1"></canvas>

<div class="hint">Используйте кнопки Вверх / Вниз, чтобы управлять прозрачностью объекта</div>

</div>

</body>

</html>

Извлечение инициализации объекта, смотрите:

<script>

//var obj = new cube();

//var obj = new sphere(6);

var obj = new sphere(16);

</script>

Это означает, что если нам нужно вывести куб — вы должны раскомментировать первую строку, если вы хотите отобразить сферу с 6 граней — нужно выбрать второй вариант.

К началу

JavaScript

Есть три файла JavaScript (main.js, meshes.js и transform.js), мы публикуем два из них, третий (transform.js) содержит только функций вычислений, связанных с вращением, масштабированием и объектов проекта. Итак, давайте рассмотрим код первого JavaScript:

Файл js/meshes.js

// Получить случайный цвет

function getRandomColor() {

var letters = '0123456789ABCDEF'.split('');

var color = '#';

for (var i = 0; i < 6; i++ ) {

color += letters[Math.round(Math.random() \* 15)];

}

return color;

}

// Подготовка объекта

function prepareObject(o) {

o.colors = new Array();

// Составим норм

o.normals = new Array();

for (var i = 0; i < o.faces.length; i++) {

o.normals[i] = [0, 0, 0];

o.colors[i] = getRandomColor();

}

// Составим центр: рассчитать максимальные позиции

o.center = [0, 0, 0];

for (var i = 0; i < o.points.length; i++) {

o.center[0] += o.points[i][0];

o.center[1] += o.points[i][1];

o.center[2] += o.points[i][2];

}

// Составим расстояния

o.distances = new Array();

for (var i = 1; i < o.points.length; i++) {

o.distances[i] = 0;

}

// Вычисляем среднее положение центра

o.points\_number = o.points.length;

o.center[0] = o.center[0] / (o.points\_number - 1);

o.center[1] = o.center[1] / (o.points\_number - 1);

o.center[2] = o.center[2] / (o.points\_number - 1);

o.faces\_number = o.faces.length;

o.axis\_x = [1, 0, 0];

o.axis\_y = [0, 1, 0];

o.axis\_z = [0, 0, 1];

}

// Объект куб

function cube() {

// Подготовим точки и граней куба

this.points=[

[0,0,0],

[100,0,0],

[100,100,0],

[0,100,0],

[0,0,100],

[100,0,100],

[100,100,100],

[0,100,100],

[50,50,100],

[50,50,0],

];

this.faces=[

[0,4,5],

[0,5,1],

[1,5,6],

[1,6,2],

[2,6,7],

[2,7,3],

[3,7,4],

[3,4,0],

[8,5,4],

[8,6,5],

[8,7,6],

[8,4,7],

[9,5,4],

[9,6,5],

[9,7,6],

[9,4,7],

];

prepareObject(this);

}

// Объект сфера

function sphere(n) {

var delta\_angle = 2 \* Math.PI / n;

// Составим вершины (точек) сферы

var vertices = [];

for (var j = 0; j < n / 2 - 1; j++) {

for (var i = 0; i < n; i++) {

vertices[j \* n + i] = [];

vertices[j \* n + i][0] = 100 \* Math.sin((j + 1) \* delta\_angle) \* Math.cos(i \* delta\_angle);

vertices[j \* n + i][1] = 100 \* Math.cos((j + 1) \* delta\_angle);

vertices[j \* n + i][2] = 100 \* Math.sin((j + 1) \* delta\_angle) \* Math.sin(i \* delta\_angle);

}

}

vertices[(n / 2 - 1) \* n] = [];

vertices[(n / 2 - 1) \* n + 1] = [];

vertices[(n / 2 - 1) \* n][0] = 0;

vertices[(n / 2 - 1) \* n][1] = 100;

vertices[(n / 2 - 1) \* n][2] = 0;

vertices[(n / 2 - 1) \* n + 1][0] = 0;

vertices[(n / 2 - 1) \* n + 1][1] = -100;

vertices[(n / 2 - 1) \* n + 1][2] = 0;

this.points = vertices;

// Составим первый слой

var faces = [];

for (var j = 0; j < n / 2 - 2; j++) {

for (var i = 0; i < n - 1; i++) {

faces[j \* 2 \* n + i] = [];

faces[j \* 2 \* n + i + n] = [];

faces[j \* 2 \* n + i][0] = j \* n + i;

faces[j \* 2 \* n + i][1] = j \* n + i + 1;

faces[j \* 2 \* n + i][2] = (j + 1) \* n + i + 1;

faces[j \* 2 \* n + i + n][0] = j \* n + i;

faces[j \* 2 \* n + i + n][1] = (j + 1) \* n + i + 1;

faces[j \* 2 \* n + i + n][2] = (j + 1) \* n + i;

}

faces[j \* 2 \* n + n - 1] = [];

faces[2 \* n \* (j + 1) - 1] = [];

faces[j \* 2 \* n + n - 1 ][0] = (j + 1) \* n - 1;

faces[j \* 2 \* n + n - 1 ][1] = (j + 1) \* n;

faces[j \* 2 \* n + n - 1 ][2] = j \* n;

faces[2 \* n \* (j + 1) - 1][0] = (j + 1) \* n - 1;

faces[2 \* n \* (j + 1) - 1][1] = j \* n + n;

faces[2 \* n \* (j + 1) - 1][2] = (j + 2) \* n - 1;

}

for (var i = 0; i < n - 1; i++) {

faces[n \* (n - 4) + i] = [];

faces[n \* (n - 3) + i] = [];

faces[n \* (n - 4) + i][0] = (n / 2 - 1) \* n;

faces[n \* (n - 4) + i][1] = i;

faces[n \* (n - 4) + i][2] = i + 1;

faces[n \* (n - 3) + i][0] = (n / 2 - 1) \* n + 1;

faces[n \* (n - 3) + i][1] = (n / 2 - 2) \* n + i + 1;

faces[n \* (n - 3) + i][2] = (n / 2 - 2) \* n + i;

}

faces[n \* (n - 3) - 1] = [];

faces[n \* (n - 2) - 1] = [];

faces[n \* (n - 3) - 1][0] = (n / 2 - 1) \* n;

faces[n \* (n - 3) - 1][1] = n - 1;

faces[n \* (n - 3) - 1][2] = 0;

faces[n \* (n - 2) - 1][0] = (n / 2 - 1) \* n + 1;

faces[n \* (n - 2) - 1][1] = (n / 2 - 2) \* n;

faces[n \* (n - 2) - 1][2] = (n / 2 - 2) \* n + n - 1;

this.faces=faces;

prepareObject(this);

}

В самом начале, мы должны подготовить все точки и фасад наших объектов. Есть 2-ве функции: куб (который генерирует начальные массивы для простого объекта куба) и сферы (для генерации сферы). Гораздо сложнее вычислить все точки и фасады для многогранной сферы. Как только мы получим все эти точки и поверхности, мы должны вычислить другие параметры (например, расстояние, абсолютный центр и три оси).

// Внутренние переменные

var canvas, ctx;

var vAlpha = 0.5;

var vShiftX = vShiftY = 0;

var distance = -700;

var vMouseSens = 0.05;

var iHalfX, iHalfY;

// Инициализация

function sceneInit() {

// Подготовка холста и контекст объектов

canvas = document.getElementById('scene');

ctx = canvas.getContext('2d');

iHalfX = canvas.width / 2;

iHalfY = canvas.height / 2;

// Начальный масштаб и перевод

scaleObj([3, 3, 3], obj);

translateObj([-obj.center[0], -obj.center[1], -obj.center[2]],obj);

translateObj([0, 0, -1000], obj);

// Подключить обработчику событий

document.onkeydown = handleKeydown;

canvas.onmousemove = handleMousemove;

// Основной цикл

setInterval(drawScene, 25);

}

// Обработчик событий OnKeyDown

function handleKeydown(e) {

kCode = ((e.which) || (e.keyCode));

switch (kCode) {

case 38: vAlpha = (vAlpha <= 0.9) ? (vAlpha + 0.1) : vAlpha; break; // Up key

case 40: vAlpha = (vAlpha >= 0.2) ? (vAlpha - 0.1) : vAlpha; break; // Down key

}

}

// Обработчик события OnMouseMove

function handleMousemove(e) {

var x = e.pageX - canvas.offsetLeft;

var y = e.pageY - canvas.offsetTop;

if ((x > 0) && (x < canvas.width) && (y > 0) && (y < canvas.height)) {

vShiftY = vMouseSens \* (x - iHalfX) / iHalfX;

vShiftX = vMouseSens \* (y - iHalfY) / iHalfY;

}

}

// Рисуем основную функцию

function drawScene() {

// Очистить холст

ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);

// Установить цвет заливки, цвет обводки, толщину линии и глобальной альфа-

ctx.strokeStyle = 'rgb(0,0,0)';

ctx.lineWidth = 0.5;

ctx.globalAlpha= vAlpha;

// Вертикальный и горизонтальный поворот

var vP1x = getRotationPar([0, 0, -1000], [1, 0, 0], vShiftX);

var vP2x = getRotationPar([0, 0, 0], [1, 0, 0], vShiftX);

var vP1y = getRotationPar([0, 0, -1000], [0, 1, 0], vShiftY);

var vP2y = getRotationPar([0, 0, 0], [0, 1, 0], vShiftY);

rotateObj(vP1x, vP2x, obj);

rotateObj(vP1y, vP2y, obj);

// Пересчитываем расстояния

for (var i = 0; i < obj.points\_number; i++) {

obj.distances[i] = Math.pow(obj.points[i][0],2) + Math.pow(obj.points[i][1],2) + Math.pow(obj.points[i][2], 2);

}

// Подготовить массив с фасадом треугольников (с расчетом максимальное расстояние для каждой грани)

var iCnt = 0;

var aFaceTriangles = new Array();

for (var i = 0; i < obj.faces\_number; i++) {

var max = obj.distances[obj.faces[i][0]];

for (var f = 1; f < obj.faces[i].length; f++) {

if (obj.distances[obj.faces[i][f]] > max)

max = obj.distances[obj.faces[i][f]];

}

aFaceTriangles[iCnt++] = {faceVertex:obj.faces[i], faceColor:obj.colors[i], distance:max};

}

aFaceTriangles.sort(sortByDistance);

// Подготовка массива с прогнозируемым пунктов

var aPrjPoints = new Array();

for (var i = 0; i < obj.points.length; i++) {

aPrjPoints[i] = project(distance, obj.points[i], iHalfX, iHalfY);

}

// Нарисовать объект (поверхность)

for (var i = 0; i < iCnt; i++) {

ctx.fillStyle = aFaceTriangles[i].faceColor;

// Начало пути

ctx.beginPath();

// Фасад индекс вершины

var iFaceVertex = aFaceTriangles[i].faceVertex;

// Переместить в исходное положение

ctx.moveTo(aPrjPoints[iFaceVertex[0]][0], aPrjPoints[iFaceVertex[0]][1]);

// И нарисовать три линии (построить треугольник)

for (var z = 1; z < aFaceTriangles[i].faceVertex.length; z++) {

ctx.lineTo(aPrjPoints[iFaceVertex[z]][0], aPrjPoints[iFaceVertex[z]][1]);

}

// Закрыть путь, и заполнить треугольник

ctx.closePath();

ctx.stroke();

ctx.fill();

}

}

// Функция сортировки

function sortByDistance(x, y) {

return (y.distance - x.distance);

}

// Инициализация

if (window.attachEvent) {

window.attachEvent('onload', sceneInit);

} else {

if (window.onload) {

var curronload = window.onload;

var newonload = function() {

curronload();

sceneInit();

};

window.onload = newonload;

} else {

window.onload = sceneInit;

}

}

Как только страница загружена, мы делаем основную инициализацию (функция sceneInit). Мы создаем canvas и контекст объектов, значит мы выполняем начальный масштаб и переводим наши объекты, которые создали в самом начале (куб или шар). Затем мы придаем обработчикам событий OnKeyDown и OnMouseMove и установим таймер, чтобы сделать нашу основную сцену (функция DrawScene). Не забывайте, что мы можем изменить globalAlpha параметры с нажатием клавиш Вверх/Вниз.

Статья подготовлена для вас коллективом сайта htmlhook.ru

Оригинал статьи: http://www.script-tutorials.com/triangle-mesh-for-3d-objects-in-html5

Перевел: Виктор Клим

Правила использования материалов сайта!

29.10.2012

21.11.2016

автор

Виктор Клим

JavaScript

canvas, webgl

Записи по метке: canvas

Анимация фона

Canvas и SVG

GPU для canvas

HTML5 Canvas Записи по метке: webgl

Анимация фона

Добавить комментарий

Ваш e-mail не будет опубликован. Обязательные поля помечены \*

Комментарий

</>

img

Имя \*

E-mail \*

КомментарииВиктор Клим: Привет! Вот как сделать значок гамбургер. Вот го...

Максат: Здравствуйте! как поставить значок гамбургера посл...

Миша: Как передать данные такого типа: у меня отображают...

Виктор Клим: Ищите в интернете документацию о fancybox modal, к...

Александр: все равно не работает( hvatay-skidku.ru/test/test...

Виктор Клим: Добавить кнопку печать в окно: <button type=...

Александр: Здравствуйте! Подскажите, а как можно вывести на ...

Свежие записи

Оператор if else

Выполнить PHP в HTML

DOM API

Всплывающая подсказка

Прокрутка страницы

Боковое меню

ToolTip

Готовность DOM

Использование браузеров за 2016 год

Содержание книги

Метки сайта

Графика SVG

HTML5 Canvas

Графика WebGL

Анимация

Теги HTML5

WordPress

Интерфейсы API

Фреймворки

Веб новости

Формы для сайта

Псевдоклассы

Микроразметка

Мобильные приложения

Скачать бесплатно

Примеры CSS

≡

Популярные записи

Примеры JavaScript

Всплывающее окно для сайта

jQuery DataTables

Bootstrap Modal

Разметка HTML5

position: sticky

Простое модальное окно

Пожертвуйте на развитие сайта любую сумму денег.

Яндекс: 410011349414697

WebMoney: R740890591055

RSS

ВКонтакте

Facebook

Twitter

G+

Быстрые ссылки

Карта сайта XML

Оформление WordPress

Примеры JavaScript

Псевдоклассы

Символы HTML

Частоты спутников

Страницы

Проект

Примеры jQuery

О блоге

Хотите сделать свой сайт еще лучше? На моём блоге Вы найдете готовые решения для оформления веб-страниц. Скрипты помогут Вам осуществить свои идеи в реальность.

Попробуйте! Это легко.

Котакты

2004 - 2017 © htmlhook.ru | SQL: 78 за 0,236 сек. 22.89 МБ • хостинг. Рекламодателям