Глава 2

Элемент Canvas

В этой главе исследуются возможности HTML5 Canvas API — мощного программного интерфейса, позволяющего генерировать и визуализировать графику, диаграммы, статические изображения и анимацию. Вы научитесь использовать основные функции визуализации для создания рисунков, способных масштабироваться и адаптироваться к среде браузера. Также будет описана методика создания динамических изображений, формируемых пользователем с помощью элемента ввода, реализованного в виде теплокарты. И конечно же, мы предупредим вас обо всех трудностях, с которыми вы можете столкнуться в процессе использования программного интерфейса HTML5 Canvas, и поделимся секретами их преодоления.

Для понимания материала, изложенного в этой главе, вполне хватит даже самого небольшого опыта работы с графикой, поэтому ничего не бойтесь и смело приступайте к освоению одного из наиболее мощных средств HTML.

Обзор средств HTML5 Canvas

Об использовании программного интерфейса HTML5 Canvas можно было бы написать целую книгу (причем весьма внушительного объема). Поскольку в нашем распоряжении имеется всего одна глава, далее обсуждаются лишь те функции, которые (по нашему мнению) используются наиболее часто.

Предыстория

Идея элемента `canvas` (холст) была впервые выдвинута и реализована компанией Apple в продукте Mac OS X WebKit для создания мини-приложений — так называемых *виджетов*. До появления этого элемента использование функций рисования в браузерах было возможно лишь при условии привлечения подключаемых модулей, таких как Flash-плагины Adobe, языка векторной разметки (VML; только в Internet Explorer) или же нестандартных программных решений на основе JavaScript.

Попробуйте, например, нарисовать обычную наклонную линию без использования элемента `canvas`. Несмотря на кажущуюся простоту этой задачи, без доступа к библиотеке функций для рисования 2D-графики сделать это довольно трудно. Именно такую библиотеку и предоставляет элемент `canvas` в HTML5, а поскольку иметь такую библиотеку в браузере чрезвычайно полезно, этот элемент был включен в спецификацию HTML5.

Ранее компания Apple намекала на возможность того, что она сохранит за собой авторские права на рабочий проект спецификации `canvas`, подготовленный группой WHATWG, что несколько обеспокоило некоторых сторонников веб-стандартизации. Однако Apple в конечном счете раскрыла свои патенты на предложенных консорциумом W3C условиях патентного лицензирования без сохранения авторских прав.

Сравнение возможностей формата SVG и элемента `canvas`

Говорит Питер: "Существенно то, что `canvas` — исключительно *растровый* холст, и поэтому нарисованные на нем изображения, в отличие от векторных изображений в формате SVG (Scalable Vector Graphics), невозможно масштабировать без потери качества. Кроме того, созданные на холсте изображения не являются частью DOM-модели страницы или какого-либо пространства имен, что иногда рассматривается как их недостаток. С другой стороны, SVG-изображения свободно масштабируются при различных величинах разрешения и позволяют точно определять, в каком месте изображения был выполнен щелчок.

Почему же тогда спецификация WHATWG HTML5 не использует исключительно SVG? Несмотря на очевидные недостатки HTML5 Canvas, в пользу этого программного интерфейса говорят следующие два факта: Элемент `canvas` обеспечивает хорошую производительность в силу того, что не должен хранить объекты для каждого рисуемого примитива и сравнительно легко реализуется на основе многих популярных библиотек для создания 2D-графики, которые существуют в других языках программирования. В конце концов, лучше синица в руке, чем журавль в небе."

Что такое холст

Когда вы используете элемент `canvas` на своей странице, он создает на ней прямоугольную область. По умолчанию ширина этой области составляет 300 пикселей, а высота — 150, но при желании для элемента `canvas` можно задать другие размеры и указать другие атрибуты. В листинге 2.1 представлен простейший элемент `canvas`, который можно добавить на HTML-страницу.

Листинг 2.1. Базовый элемент `canvas`

```
<canvas></canvas>
```

Включенным в страницу элементом `canvas` можно как угодно манипулировать, используя JavaScript. В него можно добавлять графику, линии и текст, вы можете сами рисовать на нем и даже добавлять в него полноценную анимацию.

Программный интерфейс HTML5 Canvas поддерживает те же операции 2D-рисования, что и большинство современных операционных систем и фреймворков. Если на протяжении последних лет вам приходилось заниматься программированием 2D-графики, то, работая с HTML5 Canvas, вы, скорее всего, будете чувствовать себя уверенно, поскольку спецификация специально спроектирована так, чтобы напоминать существующие системы. Если же такого опыта у вас нет, то вам предстоит узнать, насколько более мощной может быть система визуализации по сравнению с теми трюками с изображениями и CSS, которые разработчики годами использовали для создания веб-графики.

Чтобы программным путем использовать холст, прежде всего необходимо получить его *контекст* (context). После этого можно выполнить необходимые действия по отношению к контексту, а затем окончательно применить к нему их результат. Выполнение изменений на холсте аналогично выполнению транзакций в базах данных: сначала вы подготавливаете некоторые действия, а затем финализируете транзакцию.

Координаты холста

Как показано на рис. 2.1, координаты на холсте отсчитываются от точки с координатами `x=0, y=0`, находящейся в левом верхнем углу, которую мы будем называть *началом координат*, или *началом отсчета*. Координаты увеличиваются в горизонтальном направлении вдоль оси **x** и вертикальном — вдоль оси **y** (численные значения координат выражаются в пикселях).

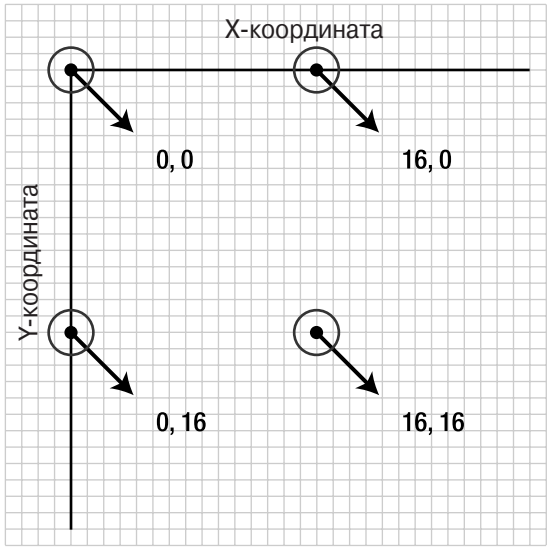

*Рис. 2.1. Расположение начала отсчета координат **x** и **y** на холсте*

Когда не следует использовать элемент `canvas`

Как бы ни был замечателен и полезен элемент `canvas`, его *не следует* использовать, если для решения задачи вполне подойдут другие элементы. Например, нельзя назвать хорошей идеей динамическое рисование всех заголовков HTML-документа на холсте, поскольку для этого достаточно использовать стили заголовков (H1, H2 и т.д.), специально предназначенные для этой цели.

Альтернативное содержимое

Если доступ к вашей веб-странице осуществляется браузером, который не поддерживает элемент `canvas` или некоторое подмножество средств HTML5 Canvas, то не помешает предоставить альтернативное содержимое (более подробно о поддержке браузерами элемента `canvas` см. табл. 2.1). Например, можно предоставить альтернативное изображение или некоторый текст, поясняющий, что именно смог бы увидеть пользователь на этом месте, если бы использовал современный браузер. В листинге 2.2 показано, как можно задать альтернативный (замещающий) текст внутри элемента `canvas`. Браузеры, не поддерживающие элемент `canvas`, могут просто выводить это альтернативное содержимое.

Листинг 2.2. Использование замещающего текста в элементе `canvas`

```
<canvas>
  Для отображения элемента canvas обновите браузер!
</canvas>
```

Вместо текста можно также указать изображение, которое будет выводиться, если поддержка элемента `canvas` в браузере отсутствует.

А что можно сказать относительно средств элемента `canvas`, предназначенных для людей с ограниченными возможностями?

Говорит Питер: "В связи с предоставлением альтернативных изображений или текста возникают вопросы доступности альтернативных средств для людей с *ограниченными возможностями* — область, в которой спецификация HTML5 Canvas, к сожалению, еще страдает заметными недостатками. Например, не существует собственного метода, обеспечивающего вставку в элемент `canvas` альтернативного текста вместо изображений, равно как не существует и метода, который предоставлял бы альтернативный текст, совпадающий с текстом, генерируемым с помощью текстовых функций Canvas API. На момент написания книги удачного решения проблем доступности средств элемента `canvas` для людей с ограниченными возможностями в случае динамически генерируемого содержимого не существовало, но над этим усиленно работает специально созданная группа. Будем надеяться, что со временем ситуация исправится."

CSS и элемент `canvas`

Как и в случае большинства элементов HTML, для добавления рамок, пробелов, полей и т.п. в элемент `canvas` можно использовать CSS-стили. Кроме того, некоторые атрибуты CSS наследуются содержимым `canvas`. Хорошим примером этого могут служить шрифты, поскольку по умолчанию к шрифтам, рисуемым на холсте, применяются настройки, используемые для самого элемента `canvas`.

Свойства, устанавливаемые для контекста, используемого в операциях с элементом `canvas`, подчиняются синтаксису, который, возможно, уже знаком вам по опыту работы с CSS. Например, для задания цветов и шрифтов в контексте используется та же нотация, что и в любом другом HTML- или CSS-документе.

Поддержка спецификации HTML5 Canvas браузерами

В настоящее время поддержка спецификации HTML5 Canvas предоставляется во всех браузерах, кроме Internet Explorer. Однако имеются некоторые части спецификации, добавленные позже, которые не имеют столь широкой поддержки. В качестве примера можно привести текстовые функции Canvas API. Однако в целом разработка спецификации продвинулась достаточно далеко, и вероятность внесения в нее существенных изменений невелика. Как следует из табл. 2.1, на момент написания книги спецификация HTML5 Canvas уже поддерживалась многими браузерами.

Таблица 2.1. Поддержка спецификации HTML5 Canvas различными браузерами

Браузер	Поддержка
Chrome	Поддерживается в версии 1.0 и выше
Firefox	Поддерживается в версии 1.5 и выше
Internet Explorer	Не поддерживается
Opera	Поддерживается в версии 9.0 и выше
Safari	Поддерживается в версии 1.3 и выше

Приведенные данные свидетельствуют о том, что спецификация HTML5 Canvas поддерживается всеми браузерами, за исключением Internet Explorer. Если вы хотите использовать элементы `canvas` в Internet Explorer, воспользуйтесь открытым проектом **explorercanvas** (`http://code.google.com/p/explorercanvas`). Для этого необходимо просто убедиться в том, что текущим браузером является Internet Explorer, и включить дескриптор `script` в свои веб-страницы для запуска сценария, как показано ниже.

```
<head>
<!--[if IE]><script src="excanvas.js"></script><![endif]-->
</head>
```

Универсальная поддержка спецификации Canvas крайне желательна для разработчиков, и поэтому постоянно возникают новые проекты, целью которых является добавление этой поддержки в среду устаревших и нестандартных браузеров. Компания Microsoft заявила о поддержке программного интерфейса Canvas в Internet Explorer 9, и поэтому поддержка данной спецификации всеми основными типами веб-браузеров уже не за горами.

В силу неоднородности этой поддержки целесообразно сначала протестировать, поддерживается ли программный интерфейс HTML5 Canvas данным браузером, и только после этого использовать его. Далее будет показано, как организовать такую проверку программным путем.

Программный интерфейс HTML5 Canvas

Этот раздел посвящен подробному рассмотрению возможностей программного интерфейса HTML5 Canvas. Эти возможности будут продемонстрированы на примере создания незамысловатого логотипа в виде изображения участка леса с несколькими деревьями и удобной тропинкой, по которой вполне может проходить маршрут соревнований по марафону на пересеченной местности. Хотя наш пример и не может претендовать на получение какой-либо награды на конкурсе работ по графическому дизайну, он достаточно хорошо иллюстрирует различные возможности программного интерфейса HTML5 Canvas.

Проверка поддержки в браузере

Прежде чем использовать элемент HTML5 `canvas`, следует проверить, поддерживает ли его браузер. Это позволит предоставить замещающий текст в том случае, если в устаревшем браузере пользователя такая поддержка отсутствует. Один из способов организации подобной проверки показан в листинге 2.3.

Листинг 2.3. Проверка поддержки элемента `canvas` в браузере

```
try {
  document.createElement("canvas").getContext("2d");
  document.getElementById("support").innerHTML =
    "Ваш браузер поддерживает элемент HTML5 Canvas.";
} catch (e) {
  document.getElementById("support").innerHTML = "Ваш браузер
     не поддерживает элемент HTML5 Canvas.";
}
```

В этом примере делается попытка создать объект `canvas` и получить доступ к его контексту. Если возникает ошибка, она перехватывается, и пользователь извещается о том, что элемент `canvas` не поддерживается данным браузером. Определенный ранее на странице элемент `support` обновляется помещением в него сообщения, информирующего о наличии или отсутствии указанной поддержки.

Этот тест позволяет судить лишь о состоянии поддержки элемента `canvas` как такового. Он ничего не говорит о том, какие именно возможности элемента поддерживаются. На момент написания книги данная спецификация была устойчива и надежно поддерживалась, так что беспокоиться по этому поводу, в общем-то, нечего.

Наконец, еще раз подчеркнем, что всегда неплохо предусмотреть для элемента `canvas` вывод альтернативного содержимого, аналогично тому, как это сделано в листинге 2.3.

Добавление элемента `canvas` на страницу

Чтобы добавить на веб-страницу элемент `canvas`, не требуется больших усилий. Как это сделать, показано в листинге 2.4.

Листинг 2.4. Элемент `canvas`

```
<canvas height="200" width="200"></canvas>
```

Результирующий холст будет отображаться на странице как "невидимый" прямоугольник с размерами 200×200 пикселей. Если хотите добавить окружающую холст рамку, используйте приведенный в листинге 2.5 код, создающий обычное обрамление средствами CSS.

Листинг 2.5. Элемент `canvas`, ограниченный сплошной рамкой

```
<canvas id="diagonal" style="border: 1px solid;"
  width="200" height="200">
</canvas>
```

Обратите внимание на включение атрибута ID со значением `"diagonal"`, который упростит нахождение элемента `canvas` программным путем. Наличие подобного идентификатора весьма существенно для любого элемента `canvas`, поскольку любая полезная операция над этим элементом должна выполняться из сценария. Отсутствие атрибута ID значительно затруднит получение доступа к элементу, с которым необходимо взаимодействовать.

На рис. 2.2 иллюстрируется, как будет выглядеть в браузере элемент `canvas`, определенный в листинге 2.5.

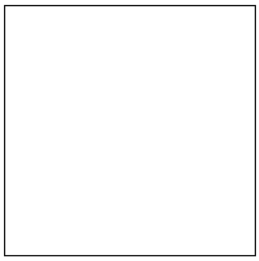

Рис. 2.2. Отображение простого элемента HTML5 `canvas` на веб-странице

Это не очень впечатляет, но любой художник скажет вам, что в этом рисунке скрыт огромный потенциал. Давайте же изобразим что-нибудь на этом девственно чистом холсте. Как уже отмечалось ранее, нарисовать на веб-странице наклонную линию, не используя возможности HTML5 Canvas, не так-то просто. Посмотрим, легко ли это сделать с помощью элемента `canvas`. Как видно из листинга 2.6, для прорисовки наклонной линии на холсте, который мы перед этим добавили в страницу, достаточно всего лишь нескольких строк кода.

Листинг 2.6. Вычерчивание наклонной линии на холсте

```
<script>
  function drawDiagonal() {
    // Получить элемент canvas и его графический контекст
    var canvas = document.getElementById('diagonal');
    var context = canvas.getContext('2d');

    // Создать путь в абсолютных координатах
    context.beginPath();
    context.moveTo(70, 140);
    context.lineTo(140, 70);

    // Прочертить линию на холсте
    context.stroke();
  }

   window.addEventListener("load", drawDiagonal, true);
</script>
```

Давайте проанализируем код JavaScript, использованный для построения наклонной линии. Несмотря на простоту этого примера, он охватывает все существенные аспекты работы с программным интерфейсом HTML5 Canvas.

Сначала осуществляется доступ к объекту `canvas`, для чего используется ссылка на значение идентификатора ID. В данном примере значением идентификатора является `"diagonal"`. Далее создается переменная `context` и вызывается метод `getContext()` объекта `canvas`, которому передается тип требуемого контекста. В данном случае передается строка `"2d"`, что означает двухмерный контекст, единственно доступный в настоящее время.

Примечание

Поддержка трехмерного (3D) контекста может появиться в одной из будущих версий спецификации.

Полученный контекст используется для выполнения операций рисования. В данном случае вычерчивается наклонная линия, что достигается вызовом трех методов, а именно `beginPath()`, `moveTo()` и `lineTo()`, с передачей координат начала и конца отрезка в качестве параметров.

Вызов методов `moveTo()` и `lineTo()` не приводит к фактическому завершению операции и прорисовке линии на холсте — для этого следует вызвать метод `context.stroke()`. Полученная наклонная линия изображена на рис. 2.3.

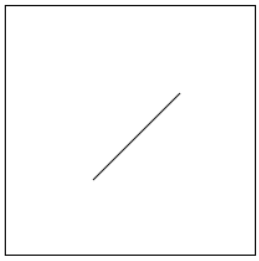

Рис. 2.3. Наклонная линия, нарисованная на холсте

Победа! Хотя усмотреть в этом результате некое предвестие революции довольно затруднительно, не следует забывать, что вычерчивание наклонной линии классическими методами HTML потребовало бы выполнения довольно сложных манипуляций, связанных с искажением изображений, а также использованием замысловатых CSS- и DOM-объектов и прочих элементов черной магии. Теперь обо всем этом можно позабыть.

Как видно из кода примера, все операции на холсте выполняются через объект контекста. То же самое можно сказать и в отношении остальных видов взаимодействия с холстом, поскольку доступ ко всем важным функциям, связанным с визуальным выводом, осуществляется лишь из контекста, а не из самого объекта `canvas`. Несмотря на то что в этой главе мы будем часто говорить о выполнении операций рисования на холсте, не забывайте: на самом деле это означает, что мы будем работать с объектом контекста, предоставляемым объектом `canvas`.

Последний пример демонстрирует, что многие операции, выполняемые над контекстом, не приводят к немедленному обновлению поверхности рисования. Выполнение таких функций, как `beginPath()`, `moveTo()` и `lineTo()`, не сопровождается мгновенным изменением внешнего вида холста. То же самое справедливо для многих функций, задающих стили, используемые на холсте, и устанавливающих предпочтительные значения параметров для него. *Путь* (path) становится видимым на экране лишь после того, как к нему будут применены методы `stroke()` или `fill()`. Иными словами, непосредственное обновление холста будет происходить лишь при выводе изображений или текста, а также при прорисовке, заполнении или очистке прямоугольников.

Использование преобразований в рисунках

Перейдем к рассмотрению другого способа рисования на холсте — *преобразованиям*. В следующем примере будет получен тот же результат, что и в предыдущем, но используемый для этого код будет другим. Если в данном случае, ввиду его простоты, еще и можно было бы возражать против привлечения преобразований на том основании, что это излишне усложняет решение, то при переходе к задачам, представляющим практический интерес, преобразования следует рассматривать как один из элементов оптимальной практики программирования, и поэтому мы будем интенсивно использовать их в оставшихся примерах. Без понимания преобразований невозможно в полной мере освоить все возможности программного интерфейса HTML5 Canvas.

Вероятно, наиболее простой способ понять, как работает система преобразований, — по крайней мере, самый простой из тех, которые не требуют привлечения устрашающих математических формул и совершения таинственных манипуляций, — это представить ее в виде модифицирующего слоя, расположенного между буфером посылаемых вами команд и выводом на экранный холст. Этот слой присутствует всегда, даже в тех случаях, когда вы не взаимодействуете с ним явно.

Модификации, или, в терминологии графических систем, *преобразования*, могут применяться последовательно, объединяться или видоизменяться по необходимости. Каждая графическая операция пропускается через модифицирующий слой, где графические объекты подвергаются всем необходимым изменениям, прежде чем выводятся на холст. И хотя общая картина при этом немного усложняется, возможности графической системы резко возрастают. Именно благодаря такому подходу стали возможными мощные преобразования, которые современное программное обеспечение для обработки изображений поддерживает в режиме реального времени, и это все в рамках библиотеки функций, сложной ровно настолько, насколько это абсолютно необходимо.

Не совершайте ошибку, полагая, что в случае отказа от использования преобразований в своем коде вы повысите его производительность. В реализации холста механизм визуализации всегда неявно использует преобразования, независимо от того, вызываете вы их непосредственно или не вызываете. Неплохо заранее представлять себе, как работает графическая система, поскольку эти знания окажутся для вас крайне важными при любой попытке выйти за рамки простейших графических операций.

Ключевая рекомендация по созданию повторно используемого кода состоит в том, что рисование, как правило, следует выполнять *с привязкой к началу координат*, а для придания рисунку окончательного вида и помещения его в нужное место на странице необходимо применять преобразования — масштабирование, трансляцию (перенос), повороты и т.п., как показано на рис. 2.4.

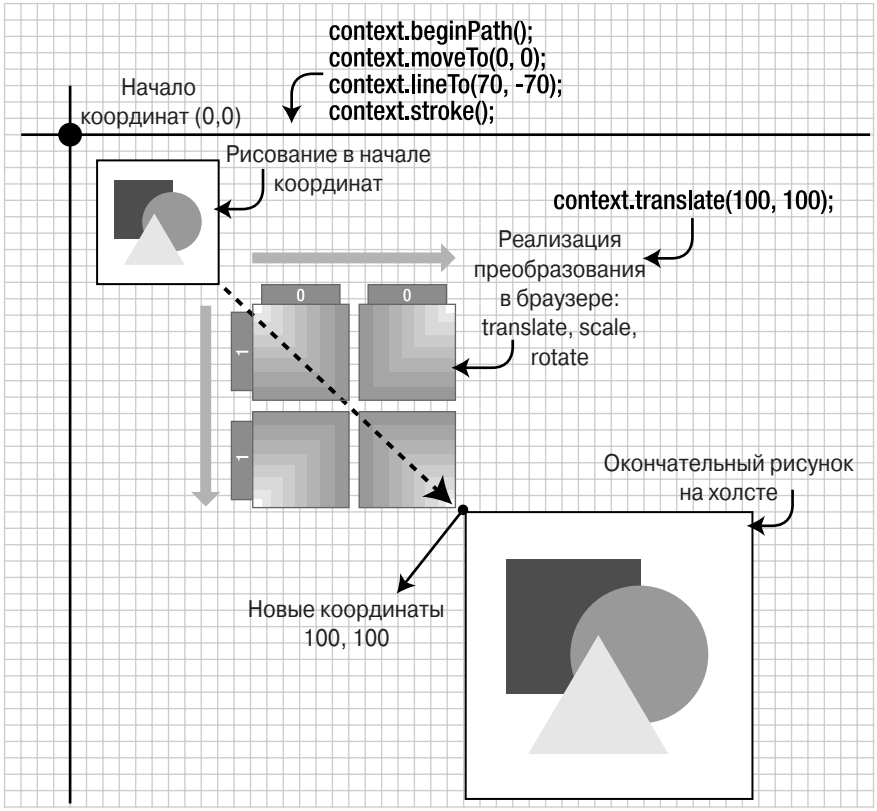

Рис. 2.4. Рисование с привязкой к началу координат и последующее преобразование результата

Применение этой рекомендации наглядно демонстрируется в листинге 2.7 на примере простейшего преобразования — `translate`.

Листинг 2.7. Создание наклонной линии на холсте с помощью трансляции

```
<script>
  function drawDiagonal() {
    var canvas = document.getElementById('diagonal');
    var context = canvas.getContext('2d');
```

```
    // Сохранить копию текущего состояния контекста
    context.save();

    // Транслировать графический контекст вправо и вниз
    context.translate(70, 140);

    // Нарисовать ту же линию, что и раньше, но с привязкой
    // к началу координат
    context.beginPath();
    context.moveTo(0, 0);
    context.lineTo(70, -70);
    context.stroke();

    // Восстановить прежнее состояние контекста
    context.restore();
  }

  window.addEventListener("load", drawDiagonal, true);
</script>
```

Давайте проанализируем сценарий JavaScript, использованный для создания второго варианта наклонной линии с помощью трансляции.

1. Прежде всего, мы получаем доступ к объекту `canvas`, ссылаясь на значение его свойства ID (в данном случае — `"diagonal"`).
2. Затем путем вызова метода `getContext()` объекта `canvas` получается значение переменной `context`.
3. Полученный контекст, который пока еще не подвергся никаким изменениям, необходимо сохранить, чтобы по завершении графических операций и преобразований можно было восстановить его исходное состояние. В противном случае внесенные изменения (трансляция, масштабирование и т.п.) будут применяться к контексту при выполнении всех будущих графический операций, что может быть нежелательным. Сохранение того состояния контекста, в котором он находился до выполнения преобразований, позволит впоследствии вновь обратиться к нему, если в этом возникнет необходимость.
4. На следующем этапе к контексту применяется трансляция. Это означает, что при визуализации любого рисунка к координатам, используемым операцией рисования (в данном случае — рисования наклонной линии), будут добавляться предоставленные вами координаты трансляции, тем самым перемещая линию в ее конечное положение, но только после того, как операция рисования будет завершена.
5. После применения трансляции контекста можно выполнить обычные графические операции по созданию наклонной линии. В данном случае эта линия создается путем вызова трех методов — `beginPath()`, `moveTo()` и `lineTo()`, которые в этот раз выполняются с привязкой к началу координат (`0,0`), а не к точке с координатами (`70,40`).
6. Сформированную таким способом линию можно визуализировать на холсте (например, путем прорисовки), вызвав для этого метод `context.stroke()`.
7. Наконец, мы восстанавливаем исходное состояние контекста для того, чтобы примененная к нему трансляция не модифицировала все последующие операции (рис. 2.5).