Лабораторная работа №4
«Расширенные инструменты контекста рисования элемента <canvas>»

Оглавление

Использование сложных цветов

Помимо однотонных цветов, холст позволяет использовать для закрашивания линий и заливок градиенты и даже выполнять заливку текстурой.

Линейный градиент

Линейный градиент создают в три этапа. Первый этап — вызов метода `createLinearGradient()` — собственно создание линейного градиента:

```
<контекст рисования>.createLinearGradient
(<Горизонтальная координата начальной точки>,
 <вертикальная координата начальной точки>,
 <горизонтальная координата конечной точки>,
 <вертикальная координата конечной точки>)
```

Координаты начальной и конечной точек градиента отсчитываются относительно холста (а не заливаемого объекта).

Метод `createLinearGradient()` возвращает объект класса `CanvasGradient`, представляющий созданный линейный градиент.

Второй этап — расстановка ключевых точек градиента. Здесь нам понадобится метод `addColorStop()` класса `CanvasGradient`:

```
<градиент>.addColorStop(<положение ключевой точки>, <цвет>)
```

Первый параметр задается в виде числа от 0.0 (начало прямой) до 1.0 (конец прямой). Результат этот метод не возвращает.

В нашем примере определяются две ключевые точки, а именно, самым первым цветом градиента указывается белый, а самым последним — черный. Затем градиент будет плавно переходить между этими цветами по всему холсту слева направо.

Третий этап — использование готового линейного градиента. Для этого представляющий его объект класса `CanvasGradient` следует присвоить свойству `strokeStyle` или `fillStyle`.

Указывая для градиента различные стартовые и конечные позиции, можно придать ему наклон для распространения в любом направлении.

Создание промежуточных ключевых точек

В градиенте можно использовать сколько угодно ключевых точек, а не только стартовую и конечную. Это позволяет дать четкое описание почти что любому типу представляемого градиентного эффекта. Для этого нужно указать процент градиента, занимаемый каждым цветом. Для распределения по градиенту стартовой позиции в формате числа с плавающей точкой берется диапазон между 0 и 1. Конечную позицию цвета вводить не нужно, поскольку она выводится из стартовой позиции следующей ключевой точки или же градиент заканчивается, если позиция является последней указанной вами.

В следующем примере создаётся эффект радуги с помощью настройки промежуточных ключевых точек. Все цвета расположены примерно на одинаковом расстоянии друг от друга (каждому цвету выделено 14 % градиента, а последнему — 16 %) но придерживаться этого не обязательно. Какие-то цвета можно прижать друг к другу, а каким-то дать больше простора. Сколько цветов использовать и где в градиенте они должны стартовать и финишировать, отдается полностью на ваше усмотрение.

Радиальный градиент

Радиальный градиент также создают в три этапа. Первый этап — вызов метода `createRadialGradient()` — создание объекта радиального градиента:

```
<контекст рисования>.createRadialGradient
(<горизонтальная координата центра внутренней окружности>,
 <вертикальная координата центра внутренней окружности>,
 <радиус внутренней окружности>,
 <горизонтальная координата центра внешней окружности>,
 <вертикальная координата центра внешней окружности>,
 <радиус внешней окружности>)
```

Параметры этого метода определяют координаты центров и радиусы обеих окружностей, описывающих радиальный градиент. Они задаются в пикселах в виде чисел.

Метод `createRadialGradient()` возвращает объект класса `CanvasGradient`, представляющий созданный нами радиальный градиент.

Второй этап — расстановка ключевых точек — выполняется с помощью уже знакомого нам метода `addColorStop()` класса `CanvasGradient`. Только в данном случае первый параметр определит относительное положение создаваемой ключевой точки на промежутке между внутренней и внешней окружностями. Он задается в виде числа от 0.0 (начало промежутка, т. е. внутренняя окружность) до 1.0 (конец промежутка, т. е. внешняя окружность).

И третий этап — использование созданного градиента.

В первом примере градиент просто начинается в центре окружности, а затем распространяется наружу. Координаты стартовой и конечной позиций те же самые, но радиус задан нулевым для стартовой позиции и охватывает весь градиент для конечной позиции.

Во втором примере градиент начинается с центра с координатами (0; 120) и радиусом 0 пикселов, а заканчивается в центре с координатами (480; 120) и радиусом 480 пикселов.

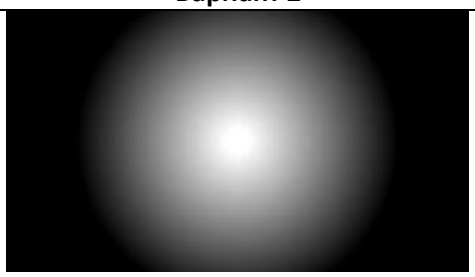
Задание для самостоятельной работы №4.1

Нарисуйте фигуру и закрасьте её с помощью линейного градиента в соответствие с вашим вариантом.

Вариант 1

Вариант 2

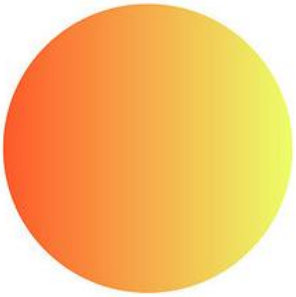
Вариант 3



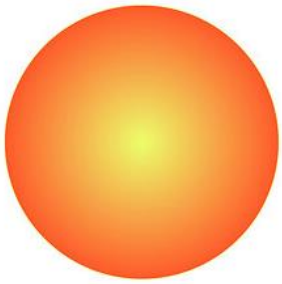
Вариант 4

Вариант 5

Вариант 6

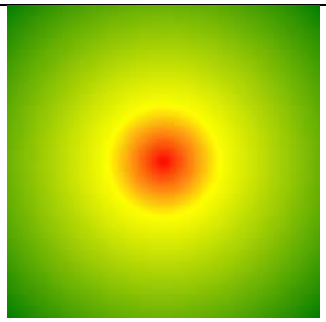


Вариант 7

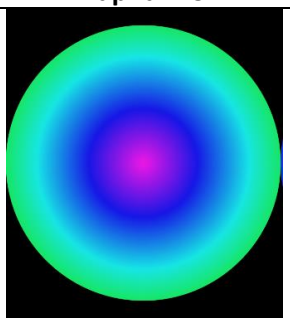


Вариант 8

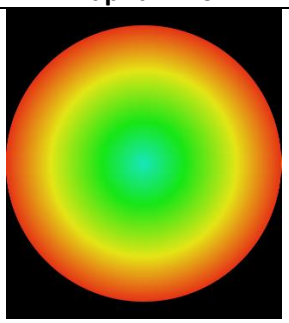
Вариант 9



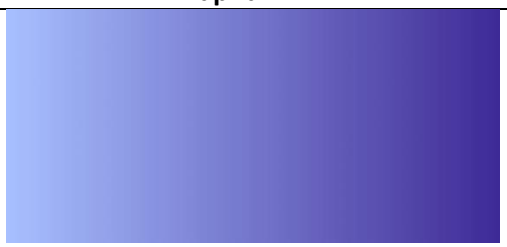
Вариант 10



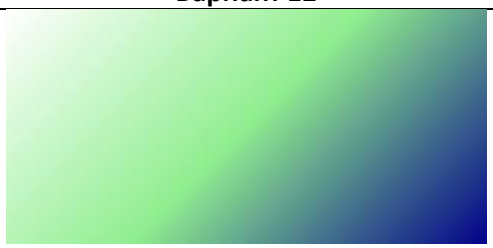
Вариант 11



Вариант 12

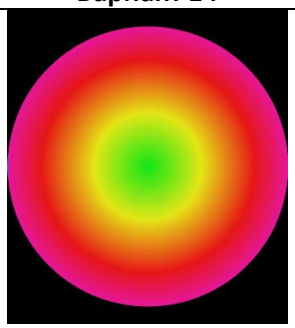


Вариант 13



Вариант 14

Вариант 15



Вариант 16

Заливка текстурой

Текстура — это обычное изображение, которым закрашиваются линии или заливки. Таким изображением может быть содержимое как графического файла, так и другого холста.

Графический цвет создают в три этапа. Первый этап необходим только в том случае, если мы используем в качестве цвета содержимое графического файла. Такой файл нужно как-то

загрузить, удобнее всего — с помощью объекта класса `Image`, который представляет графическое изображение, хранящееся в файле.

Сначала с помощью оператора `new` создаем объекта класса `Image`. Следующая инструкция прикрепляет к событию `onload` функцию, создающую повторяющийся узор для свойства `fillStyle`. Далее мы присваиваем свойству `src` этого объекта интернет-адрес загружаемого графического файла в виде строки. Все — теперь можно использовать данный экземпляр объекта `Image` для создания графического цвета.

Если бы в данном примере не использовалось событие `onload`, а вместо этого код просто выполнялся бы сразу, как только он попадался в сценарии, изображение могло быть к этому времени еще не загруженным и могло не выводиться на экран. Прикрепление к этому событию гарантирует доступность изображения для использования на холсте, поскольку событие наступает только в результате успешной загрузки изображения.

Второй этап — собственно создание графического цвета с помощью метода `createPattern`:

```
<контекст рисования>.createPattern(<графическое изображение или холст>, <режим повторения>);
```

Первый параметр задает графическое изображение в виде объекта класса `Image`, элемента `img` или объекта другого холста.

Часто бывает так, что размеры заданного графического изображения меньше, чем фигуры, к которой должен быть применен графический цвет. В этом случае изображение повторяется столько раз, чтобы полностью «вымостить» линию или заливку. Режим такого повторения задает второй параметр метода `createPattern()`. Его значение должно быть одной из следующих строк:

- `"repeat"` — изображение будет повторяться по горизонтали и вертикали;
- `"repeat-x"` — изображение будет повторяться только по горизонтали;
- `"repeat-y"` — изображение будет повторяться только по вертикали;
- `"no-repeat"` — изображение не будет повторяться никогда; в этом случае часть фигуры останется не занятой им.

Метод `createPattern()` возвращает объект класса `CanvasPattern`, представляющий созданный нами графический цвет.

Третий этап — использование готового графического цвета — выполняется так же, как для градиентов.

В нашем примере мы загружаем графический файл graphic_color.jpg, создаем на его основе повторяющийся по горизонтали и вертикали графический цвет и рисуем с его помощью прямоугольник.

Узор заливки применяется по отношению ко всей области холста, поэтому, когда команда заливки настроена на применение только к меньшей по размеру области внутри холста, изображения выводятся обрезанными.

Создание тени

Холст позволяет создавать тень у всех рисуемых фигур. Для задания ее параметров применяют четыре свойства:

- `shadowOffsetX` — смещение тени по горизонтали относительно фигуры (в виде числа в пикселах, значение по умолчанию — 0). При положительном значении тень смещается вправо, а при отрицательном – влево;
- `shadowOffsetY` — смещение тени по вертикали относительно фигуры (в виде числа в пикселах, значение по умолчанию — 0). При положительном значении тень смещается вниз, а при отрицательном – вверх;
- `shadowBlur` — степень размытия тени в виде числа: чем больше это число, тем сильнее размыта тень (значение по умолчанию — 0, т. е. отсутствие размытия);
- `shadowColor` — цвет тени (по умолчанию — черный непрозрачный). Если применяется размытие, этот цвет будет в размываемой области смешиваться с фоном.

Задание для самостоятельной работы №4.2

Добавьте тень к рисунку, который вы нарисовали в самостоятельном задании №4.1.

Управление наложением графики

Когда мы рисуем какую-либо фигуру поверх уже существующей, новая фигура накладывается на старую, перекрывая ее. Это поведение холста по умолчанию, которое мы можем изменить, указав другие значения для свойства `globalCompositeOperation`. Рассмотрим следующие строковые значения этого свойства:

1. `"source-over"` — новая фигура накладывается на старую, перекрывая ее (значение по умолчанию);
2. `"source-atop"` — отображается только та часть новой фигуры, которая накладывается на старую; остальная часть новой фигуры не выводится. Старая фигура выводится целиком и находится ниже новой;
3. `"source-in"` — отображается только та часть новой фигуры, которая накладывается на старую. Остальные части новой и старой фигур не выводятся;
4. `"source-out"` — отображается только та часть новой фигуры, которая не на кладывается на старую. Остальные части новой фигуры и вся старая фигура не выводятся;
5. `"destination-over"` — новая фигура перекрывается старой;
6. `"destination-atop"` — отображается только та часть старой фигуры, которая накладывается на новую; остальная часть старой фигуры не выводится. Новая фигура выводится целиком и находится ниже старой;
7. `"destination-in"` — отображается только та часть старой фигуры, на которую накладывается новая. Остальные части новой и старой фигур не выводятся;
8. `"destination-out"` — отображается только та часть старой фигуры, на которую не накладывается новая. Остальные части новой фигуры и вся старая фигура не выводятся;
9. `"lighter"` — цвета накладывающихся частей старой и новой фигур складывают ся, результирующий цвет получается более светлым;
10. `"xor"` — отображаются только те части старой и новой фигур, которые не накладываются друг на друга;
11. `"copy"` — выводится только новая фигура; все старые фигуры удаляются с холста.
12. `"difference"` — вычитает цвет старой фигуры из новой — или наоборот — чтобы всегда получать положительное значение.
13. `"multiply"` — цвет новой фигуры умножается на цвет старой фигуры. В результате получается более темное изображение.

14. "screen" — цвет новой фигуры инвертируется, умножается на цвет старой фигуры и снова инвертируются. В результате получается более светлое изображение (противоположность "multiply").
15. "darken" – выбирается более темный цвет накладывающихся фигур.
16. "lighten" – выбирается более светлый цвет накладывающихся фигур.

Заданный нами способ наложения действует только для графики, которую мы нарисуем после этого. На уже нарисованную графику он не влияет.

Задание для самостоятельной работы №4.3

Запустите код, который рисует два накладывающихся прямоугольника разных цветов и позволяет изучать поведение холста при разных значениях свойства globalCompositeOperation. Измените значение этого свойства в соответствии с вашим вариантом, перезагрузите страницу нажатием клавиши <F5> и посмотрите, что получится.

Использование масок

Маской называется особая фигура, задающая своего рода "окно", сквозь которое будет видна часть графики, нарисованная на холсте. Вся графика, не попадающая в это "окно", будет скрыта; при этом сама маска на хост не выводится.

Ниже перечислены действия, необходимые для создания маски.

1. Рисуем сложный контур, который станет маской.
2. Обязательно делаем его замкнутым.
3. Вместо вызова методов stroke или fill вызываем метод clip. Этот метод не принимает параметров и не возвращает результат.
4. Рисуем графику, которая будет находиться под маской.

После этого нарисованная нами на шаге 4 графика будет частично видна сквозь маску.

Приведенный код рисует маску в виде треугольника, а потом рисует прямоугольник. Часть этого прямоугольника будет видна сквозь маску.

Работа с отдельными пикселами

Мы можем работать с отдельными пикселами графики, нарисованной на холсте. Это может пригодиться при создании очень сложного изображения или при наложении на графику специальных эффектов наподобие размытия.

Получение массива пикселов

Получить массив пикселов, представляющий нарисованную на холсте графику или ее фрагмент, позволяет метод getImageData(). Формат метода:

```
<контекст рисования>.getImageData(<x>, <y>, <Ширина>, <Высота>)
```

Первые два параметра указывают координату левого верхнего угла фрагмента нарисованной графики, два последних — его ширину и высоту.

Метод getImageData() возвращает объект класса ImageData, представляющий массив пикселов, которые составляют часть сцены с указанными нами параметрами:

```
const idSample = ctxCanvas.getImageData(200, 150, 100, 100);
```

Создание пустого массива пикселов

Чтобы самостоятельно нарисовать какое-либо изображение, можно сначала создать пустой массив пикселов, вызвав метод `createImageData()`:

```
<контекст рисования>.createImageData(<Ширина>, <Высота>)
```

Метод возвращает объект класса `ImageData`:

```
const idEmpty = ctxCanvas.createImageData(400, 300);
```

Здесь мы создаем пустой массив пикселов тех же размеров, что и сам холст.

Манипуляция пикселами

Теперь мы можем начать работать с отдельными пикселами полученного массива, задавая для них цвет и значения полупрозрачности и тем самым формируя какое-либо изображение или изменяя уже существующее.

Класс `ImageData` поддерживает свойство `data`. Его значением является объект-коллекция, хранящая набор чисел:

- первое число представляет собой долю красного цвета в цвете первого пиксела массива;
- второе число — долю зеленого цвета в цвете первого пиксела;
- третье число — долю синего цвета в цвете первого пиксела;
- четвертое число — степень полупрозрачности цвета первого пиксела;
- пятое число — долю красного цвета в цвете второго пиксела;
- шестое число — долю зеленого цвета в цвете второго пиксела;
- седьмое число — долю синего цвета в цвете второго пиксела;
- восьмое число — степень полупрозрачности цвета второго пиксела;
- ...
- n-3-е число — долю красного цвета в цвете последнего пиксела;
- n-2-е число — долю зеленого цвета в цвете последнего пиксела;
- n-1-е число — долю синего цвета в цвете последнего пиксела;
- n-е число — степень полупрозрачности цвета последнего пиксела.

Все значения, включая и степень полупрозрачности, должны укладываться в диапазон от 0 до 255. Для степени полупрозрачности значение 0 задает полную прозрачность, а 255 — полную непрозрачность.

Нумерация пикселов в массиве идет слева направо и сверху вниз, т. е. по строкам.

Поскольку набор чисел, хранящийся в свойстве `data` класса `ImageData`, представляет собой коллекцию, для доступа к отдельным значениям мы можем использовать тот же синтаксис, что и в случае обычных массивов. Также мы можем воспользоваться поддерживаемым всеми коллекциями свойством `length`, возвращающим размер коллекции.

Вывод массива пикселов

Завершив формирование нового изображения в массиве пикселов, мы можем вывести его на холст, вызвав метод `putImageData()`:

```
<контекст рисования>.putImageData(<Массив пикселов>, <x1>, <y1>[,
<x2>, <y2>,<Ширина>, <Высота>])
```

Второй и третий параметры задают координату точки, где будет находиться левый верхний угол выводимого массива пикселов. Четвертый и пятый параметры задают координаты точки, где находится левый верхний угол фрагмента массива пикселов, который должен быть выведен на холст, а шестой и седьмой— ширину и высоту выводимого фрагмента. Если эти параметры не указаны, будет выведен весь массив пикселов:

```
ctxCanvas.putImageData(idEmpty, 0, 0);
```

В примере приведен код, рисующий на странице прямоугольник с градиентной заливкой. При этом цвета становятся все более и более прозрачными.