Лабораторная работа №5

«Двухмерные геометрические преобразования»

Оглавление

Введение

Теперь, после знакомства с основами рисования фигур, сделаем еще один шаг вперед и попробуем выполнить геометрические преобразования над этими моделями, которые относятся к классу **аффинных преобразований**. На лекциях мы будем рассматривать математический аппарат, описывающий каждое преобразование. Мы увидим, что такие двухмерные преобразования можно представить в виде матриц 3×3 **(матриц преобразований)**.

Преобразования модели

Как правило объекты сцены создаются в собственной **локальной** системе координат. Для их расположения и ориентации в **мировой** системе координат применяются геометрические преобразования.

Предположим, что вы пишете компьютерную игру, для которой нужен космический корабль. Сначала вы смоделируете свой космический корабль в локальных координатах. Затем преобразование модели переведет космический корабль в заданную точку в мировой системе

координат, развернет его заданным образом, а затем, возможно, увеличит его до размера, который соответствует вашей игре.

Преобразование, выполняющее переход от координат моделирования к мировым координатам называется **преобразованием модели (model transformation)**, а соответствующая ему матрица называется **модельной**.

При создании контекста рисования модельная матрица инициализируется значениями, предлагаемыми по умолчанию, при которых все операции рисования выполняются без изменений.

Сохранение и загрузка состояния

Первое, что нам нужно рассмотреть применительно к преобразованиям, — это сохранение и загрузка состояния холста. Эти возможности нам очень пригодятся в дальнейшем.

При сохранении состояния холста сохраняются:

- текущая модельная матрица;
- значения свойств `globalAlpha`, `globalCompositeOperation`, `fillStyle`, `lineCap`, `lineJoin`, `lineWidth`, `miterLimit` и `strokeStyle`;
- все заданные маски.

Метод `save` сохраняет состояние холста. Он не принимает параметров и не возвращает результат.

Состояние холста сохраняется в стеке и впоследствии оно может быть восстановлено. Более того, сохранять состояние холста можно несколько раз; при этом все предыдущие состояния остаются в памяти и их также можно восстановить.

Восстановить сохраненное ранее состояние можно вызовом метода `restore`. Он также не принимает параметров и не возвращает результат.

При вызове этого метода будет восстановлено самое последнее из сохраненных состояний холста. При последующем его вызове будет восстановлено предпоследнее сохраненное состояние и т. д. Этой особенностью часто пользуются для создания сложной графики.

В приведенном примере мы сначала назначаем свойству `fillStyle` красный цвет и вызываем метод `save()`, после чего изменяем значение `fillStyle` на зеленый цвет и еще раз вызываем метод `save()` для сохранения параметров. Затем мы назначаем свойству `fillStyle` синий цвет и рисуем прямоугольник, левый верхний угол которого отображается в точке (100,100). Первый вызов метода `restore()` восстанавливает для свойства `fillStyle` зеленый цвет, так что следующий прямоугольник рисуется зеленым цветом, но его левый верхний угол находится в точке (110,110). Второй вызов `restore()` восстанавливает первоначальное значение свойства `fillStyle`, поэтому последний прямоугольник рисуется красным цветом с левым верхним углом в точке (0,0).

Геометрические преобразования

Преобразование модели обычно является комбинацией следующих преобразований:

- перемещение;
- поворот;
- масштабирование.

Перемещение

Метод `translate` умножает текущую матрицу модели на матрицу перемещения:

```
<контекст рисования>.translate {<горизонтальная координата>,
<вертикальная координата>)
```

Рассмотрим пример, который делает следующее:

1. Сохраняет текущее состояние холста.
2. Умножает текущую матрицу модели на матрицу перемещения с компонентами [100,100].
3. Рисует красный квадрат со стороной 50 пикселов, верхний левый угол которого находится в точке начала координат.
4. Опять умножает текущую матрицу модели на матрицу перемещения с компонентами [100,100].
5. Рисует зеленый квадрат со стороной 50 пикселов, верхний левый угол которого находится в начале координат.
6. Восстанавливает сохраненное состояние холста, в том числе и модельную матрицу.
7. Рисует синий квадрат со стороной 50 пикселов, верхний левый угол которого находится в начале координат.

В результате мы увидим три цветных квадрата, расположенные на воображаемой диагонали, тянущейся от верхнего левого угла холста вниз и вправо.

Задание для самостоятельной работы №1

Напишите программу, которая рисует равносторонний треугольник. Затем переместите этот треугольник с помощью заданного вектора перемещения. Отобразите треугольник в исходном положении и после выполнения преобразования, как показано на рис. 1.

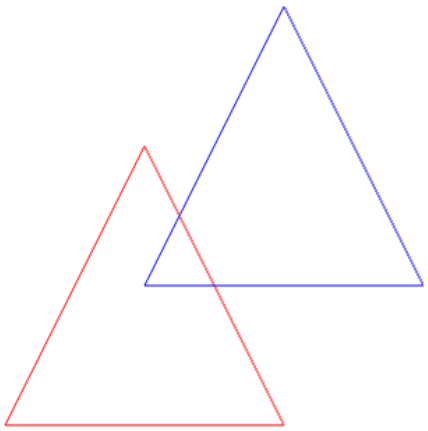

Рис. 1. Перемещение треугольника

Вращение

После знакомства с операцией перемещения перейдем к операции вращения. В своей основе, реализация вращения мало чем отличается от реализации перемещения.

Метод `rotate` умножает текущую матрицу модели на матрицу вращения:

```
<контекст рисования>.rotate (<угол поворота>)
```

Угол поворота задается в виде числа в радианах и отсчитывается от горизонтальной оси. Отметим, что положительным направлением считается направление по часовой стрелке. Вращение происходит вокруг точки начала координат.

В приведенном примере мы сначала нарисовали красный прямоугольник и переместили его на вектор [200,150]. Затем мы нарисовали синий прямоугольник, к которому сначала применили преобразование поворота на π/6 радиан (30°) и затем такое же преобразование перемещения.

Задание для самостоятельной работы №2

Написать программу, которая будет осуществлять поворот нарисованного в задании 5.1 треугольника на заданный угол вокруг одной из его вершин. Отобразите треугольник в исходном положении и после выполнения преобразования, как показано на рис. 2.

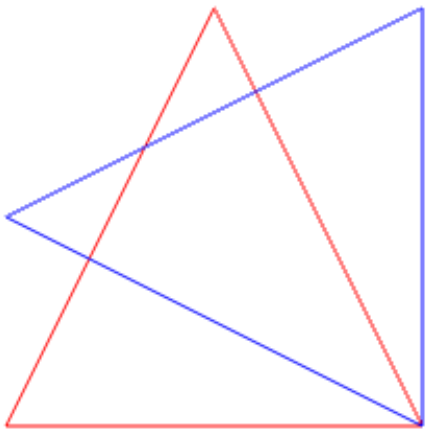

Рис. 2. Поворот треугольника

Масштабирование

Метод `scale` умножает текущую матрицу модели на матрицу масштабирования:

```
<контекст рисования>.scale(<масштаб по горизонтали>,
<масштаб по вертикали>)
```

Параметры этого метода определяют коэффициенты масштабирования, которые могут быть произвольными числами.

Код примера делает следующее:

1. Сохраняет текущее состояние холста.
2. Рисует красный квадрат со стороной 50 пикселов, верхний левый угол которого находится в начале координат.
3. Увеличивает масштаб по горизонтали в 3 раза.
4. Рисует зеленый квадрат со стороной 50 пикселов.
5. Восстанавливает сохраненное ранее состояние холста и сохраняет его снова.

6. Увеличивает масштаб по вертикали в 3 раза.
7. Рисует синий квадрат со стороной 50 пикселов.
8. Восстанавливает сохраненное ранее состояние холста и сохраняет его снова.
9. Увеличивает масштаб по обеим координатным осям в 3 раза.
10. Рисует черный квадрат со стороной 50 пикселов.
11. Восстанавливает сохраненное ранее состояние холста.

В результате мы увидим четыре прямоугольника с реальными размерами 50×50, 150×50, 50×150 и 150×150 пикселов.

Задание для самостоятельной работы №3

Написать программу, которая будет осуществлять масштабирование нарисованного в задании 5.1 треугольника относительно его центра с заданными значениями масштабных коэффициентов. Отобразите треугольник в исходном положении и после выполнения преобразования, как показано на рис. 3.

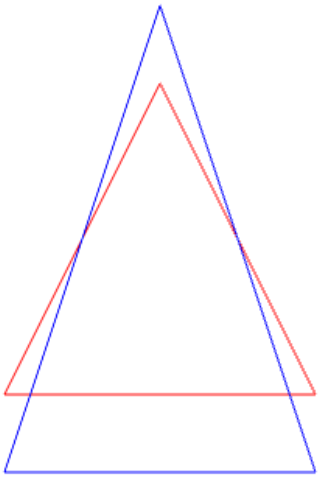

Рис. 3. Результат масштабирования треугольника по вертикали с коэффициентом 1.5

Создание произвольной матрицы преобразования

Метод `transform` умножает текущую матрицу модели на матрицу произвольного преобразования:

```
<контекст рисования>.transform(m11, m12, m21, m22, tx, ty)
```

Матрица произвольного преобразования имеет следующий вид:

$$\begin{pmatrix} m_{11} & m_{12} & t_x \\ m_{21} & m_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix}.$$

Метод `setTransform` устанавливает заданную матрицу в качестве матрицы модели:

```
<контекст рисования>.setTransform(m11, m12, m21, m22, tx, ty)
```

Этот метод можно использовать вместо методов `save` и `restore` для установки нужной матрицы модели.

В примере демонстрируется задание матрицы для преобразования сдвига. Параметр сдвига может быть любым числом.

Объединение нескольких преобразований

При выполнении нескольких преобразований, вызовы функций, формирующих матрицы отдельных преобразований, осуществляется в обратном порядке. То есть они соответствует порядку записи сомножителей при вычислении итоговой матрицы преобразования:

$$\mathbf{M} = \mathbf{M}_2 \cdot \mathbf{M}_1.$$

Задание для самостоятельной работы №4 (не обязательно)

Написать программу, которая будет осуществлять перемещение нарисованного в задании 5.1 треугольника, а затем его поворот на заданный угол вокруг исходного положения его центра. Отобразите треугольник в исходном положении и после выполнения каждого из двух преобразований.

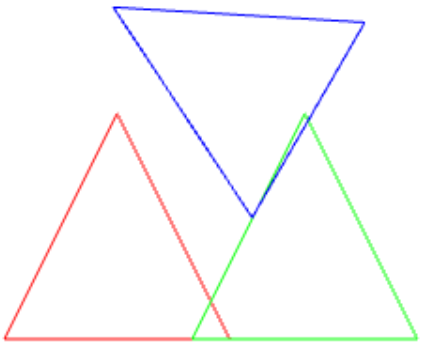

Рис. 4. Перемещение и поворот треугольника

Задание для самостоятельной работы №5 (не обязательно)

Написать программу, которая будет осуществлять поворот на заданный угол нарисованного в задании 5.1 треугольника, а затем его перемещение. Отобразите треугольник в исходном положении и после выполнения каждого из двух преобразований.

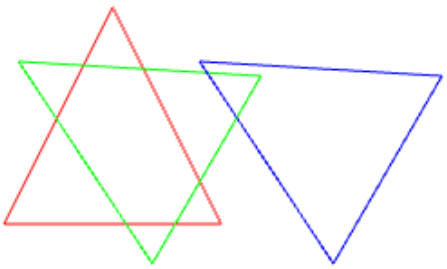

Рис. 5. Поворот и перемещение треугольника

Задание для самостоятельной работы №6

Создать анимацию аналоговых часов с вращающимися часовой, минутной и секундной стрелками.

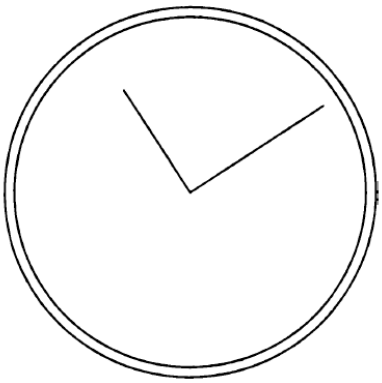

Рис. 6. Часы

Создание анимации с помощью функции `requestAnimationFrame()`

Функция `requestAnimationFrame()` вызывает функцию, указанную в первом параметре, некогда в будущем. После вызова функции необходимо вновь потребовать от браузера вызвать функцию в будущем, потому что предыдущее требование автоматически аннулируется после его выполнения.

Интервалы между моментами времени вызова функции могут быть разными, потому что нагрузка на браузер с течением времени может изменяться.

Если интервалы времени могут быть разными, следовательно, простое увеличение текущего угла поворота на постоянную величину (градусы в секунду) при каждом вызове приведет к скачкообразному изменению скорости вращения стрелок, не соответствующему их реальному положению.

Поэтому будем определять новый угол поворота исходя из интервала времени, прошедшего с момента последнего вызова. Для этого создадим переменную `g_last`, которая будет хранить время последнего вызова. Обозначим переменную-аргумент функции `now`, в которую передается текущее время.

Затем вычислим продолжительность интервала времени, прошедшего с момента предыдущего вызова, с помощью разности `now-g_last`, и сохраним результат в переменной `elapsed`. После этого, исходя из значения `elapsed`, вычислим величину угла поворота:

```
let newAngle = angle + (ANGLE_STEP * elapsed) / 1000.0;
```

Чтобы вычислить угол поворота, исходя из скорости вращения (градусы в секунду), достаточно просто умножить скорость `ANGLE_STEP` на `elapsed/1000`. Фактически же, сначала выполняется умножение на `elapsed`, а затем результат делится на 1000. Реализовано так потому, что такая последовательность операций дает чуть более точный результат, но сути это не меняет.

Наконец, нужно проверить выход величины угла поворота за границу 360 (градусов) и полученный результат использовать в функции поворота.

Задание для самостоятельной работы №7 (не обязательно)

Создать анимацию аналоговых часов из задания 5.6 с помощью функции `requestAnimationFrame()`.