Лабораторная работа №6

«Двухмерные геометрические преобразования»

Оглавление

Введение

Теперь, после знакомства с основами рисования простых фигур, таких как треугольники и прямоугольники, сделаем еще один шаг вперед и попробуем выполнить геометрические преобразования над этими моделями.

Преобразования модели

Как правило объекты сцены создаются в собственной **локальной** системе координат. Для их расположения и ориентации во внешней (**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, развернем 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**.

Преобразование модели обычно является комбинацией следующих преобразований:

- перемещение
- поворот
- масштабирование

Работа с векторами и матрицами с помощью библиотеки glMatrix

Для работы с векторами и матрицами будем использовать библиотеку glMatrix.

Перейдите на сайт http://glMatrix.net (рис. 1), скачайте, разархивируйте и сохраните файл glMatrix.js в папку lib. Будем использовать несжатую версию, поскольку ее исходный код проще читать и отлаживать.

На сайте также содержится документация по всем доступным в библиотеке методам.

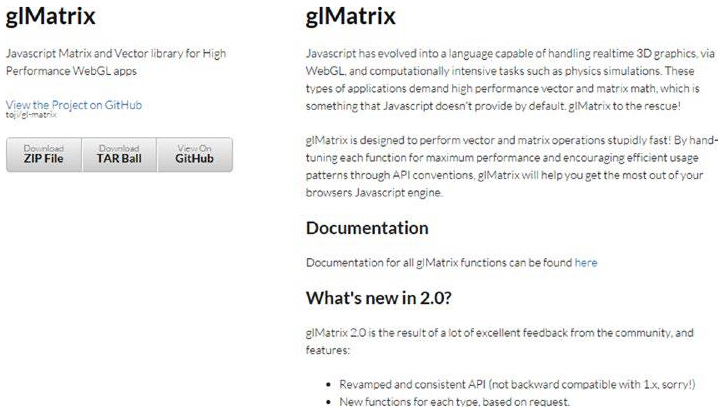

Рис. 1. Сайт библиотеки glMatrix

Применение преобразований

Перемещение

Для начала напишем программу, которая будет перемещать треугольник. За основу возьмем программу рисования треугольника из лабораторной работы №4.

Операции трансформации должны выполняться для каждой вершины, поэтому их следует реализовать в вершинном шейдере.

Так как преобразования являются жесткими (одинаковыми для всех вершин), будем использовать переменную со спецификатором `uniform` для передачи матрицы трансформации в вершинный шейдер. Матрица преобразования может иметь размер 2×2, 3×3 или 4×4 соответственно нужно указать ее тип `mat2`, `mat3` или `mat4`.

В вершинном шейдере требуется выполнить умножение матрицы преобразования на вектор с координатами вершины. В языке GLSL ES умножение матрицы на вектор можно выполнить непосредственно в одной строке кода.

В вершинный шейдер матрица 2×2, 3×3 или 4×4 передается в соответствующую `uniform`-переменную вызовом метода `gl.uniformMatrix[2-4]fv()`. Обратите внимание на

последнюю букву «v» в имени метода, которая указывает, что метод способен записать в переменную множество значений.

`gl.uniformMatrix[2-4]fv (location, transpose, array)`
Записывает матрицу, находящуюся в массиве `array`, в `uniform`-переменную `location`.

Параметры:

`location`	Определяет `uniform`-переменную, куда будет записана матрица.
`transpose`	Этот параметр определяет необходимость транспонирования матрицы. Т.к. в библиотеке glMatrix предполагается, что матрицы хранятся по столбцам, аналогично типам `mat2`, `mat3` и `mat4` в языке шейдеров, то выполнять транспонирование при передаче матрицы в шейдер не нужно.
`array`	Определяет массив с матрицей. В glMatrix матрицы по умолчанию представлены одномерными массивами типа `Float32Array`, поэтому их можно сразу использовать для передачи в шейдеры.

Возвращаемое значение: нет

Матрица перемещения задается с помощью методов `fromTranslation` классов `mat2d`, `mat3` или `mat4` библиотеки glMatrix.

После знакомства с операцией перемещения перейдем к операции вращения. В своей основе, реализация вращения мало чем отличается от реализации перемещения – она точно так же требует выполнения операций с координатами в вершинном шейдере.

Задание для самостоятельной работы

Написать программу, которая будет перемещать треугольник с помощью заданного вектора перемещения. За основу можно взять программу рисования треугольника из лабораторной работы №4.

Вращение

Пусть требуется повернуть фигуру относительно начала координат против часовой стрелки на угол β градусов.

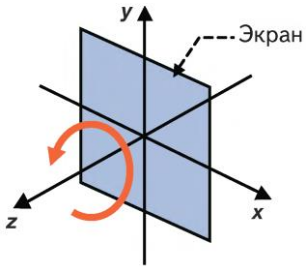

Рис. 2. Положительное вращение относительно начала координат

Если величина угла поворота β выражается положительным значением, считается, что вращение выполняется против часовой стрелки, если смотреть вдоль оси вращения в направлении отрицательных значений (см. рис. 2); такое вращение называют положительным вращением. По аналогии с системой координат, ваши руки могут помочь вам определить направление вращения. Если правой рукой изобразить систему координат так, чтобы большой палец, изображающий ось вращения, смотрел вам в лицо, согнутые пальцы будут показывать направление положительного вращения. Этот прием называют правилом вращения правой руки.

Если углу β присвоить отрицательное значение треугольник будет повернут в противоположную сторону (по часовой стрелке).

Матрица вращения задается с помощью методов `fromRotation` классов `mat2`, `mat3` или `mat4` библиотеки glMatrix.

Задание для самостоятельной работы

Написать программу, которая будет осуществлять поворот треугольника вокруг начала координат на заданный угол.

Масштабирование

Для выполнения операции масштабирования необходимо определить масштабные коэффициенты. Центром масштабирования будет считать точку – начало координат.

Матрица масштабирования задается с помощью методов `fromScaling` классов `mat2`, `mat3` или `mat4` библиотеки glMatrix.

Задание для самостоятельной работы

Написать программу, которая будет осуществлять масштабирование треугольника относительно начала координат с заданными значениями масштабных коэффициентов.

Объединение нескольких преобразований

Посмотрим, как можно использовать методы объектов `mat2`, `mat3` или `mat4` для объединения двух преобразований: перемещения с последующим вращением.

Выполним следующие два преобразования, изображенные на рис. 3:

1. Перемещение треугольника по оси *X* (1).

2. Поворот вокруг оси начала координат (2).

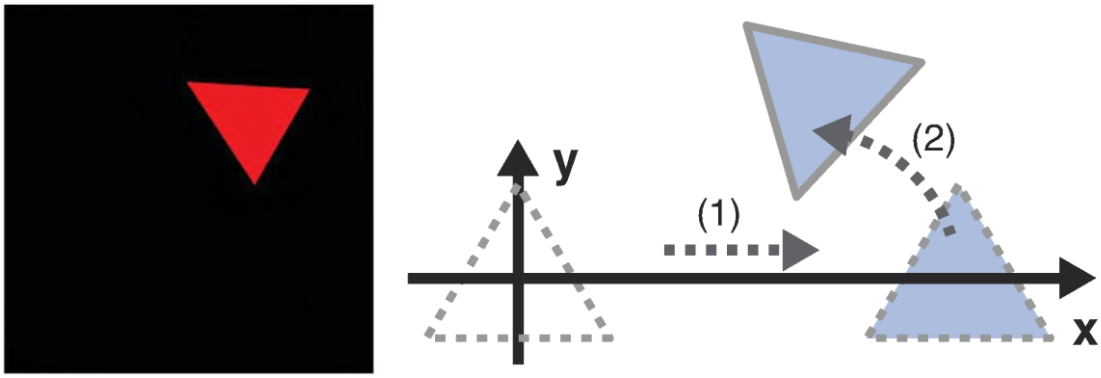

Рис. 3. Перемещение и поворот треугольника

Вызов функций, формирующий матрицы соответствующих преобразований, осуществляется в обратном порядке. То есть он соответствует порядку записи сомножителей при вычислении итоговой матрицы преобразования. При этом для первой матрицы (матрицы, формирующей последнее преобразование) используется функция с именем, начинающимся с префикса `from` (в данном случае – `fromRotation`). Затем вызываются функции с именем без этого префикса (для данного примера – `translate`). Они осуществляют умножение текущей матрицы $\mathbf{M}_2$ на матрицу заданного преобразования $\mathbf{M}_1$:

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

Затем перепишем программу так, чтобы она сначала выполняла поворот, а потом перемещение. Для этого требуется всего лишь поменять местами операции вращения и перемещения.

Результат работы этой программы показан на рис. 4.

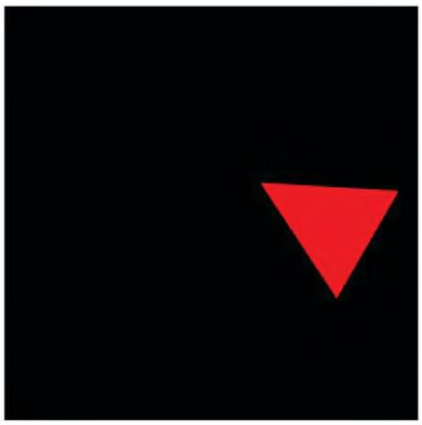

Рис. 4. Поворот и перемещение треугольника

Изменив порядок операций вращения и перемещения, получили другой результат. Причина этого станет очевидна, если посмотреть на рис. 5.

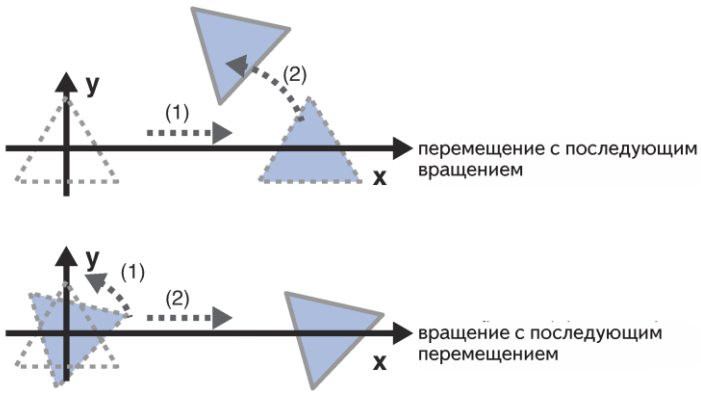

Рис. 5. Изменение порядка преобразований приводит к другому результату

Анимация

Сделаем следующий шаг и применим имеющиеся знания для реализации анимационных эффектов.

Создадим программу, которая будет непрерывно вращать треугольник с постоянной скоростью (45 градусов/сек).

Основы анимации

Чтобы воспроизвести эффект вращения треугольника, нужно просто перерисовывать треугольник, немного поворачивая его каждый раз.

На рис. 6 показаны отдельные треугольники, которые рисуются в моменты времени t0, t1, t2, t3 и t4. Каждый треугольник все еще остается всего лишь статическим изображением, но при этом все они повернуты на разные углы, по сравнению с исходным положением. Если вы увидите

последовательную смену этих изображений, ваш мозг обнаружит изменения и сгладит границы между ними, породив ощущение поворачивания треугольника. Конечно, перед рисованием нового треугольника необходимо очистить область на экране с помощью вызова функции `gl.clear()`.Такой способ анимации можно применять не только к 2-мерным фигурам, но и к трехмерным объектам.

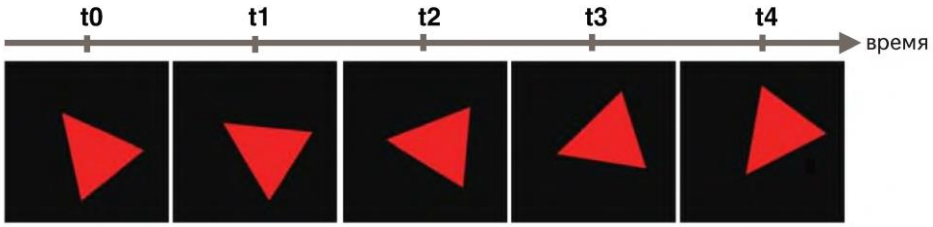

Рис. 6. Положение треугольника в различные моменты времени

Повторение вызовов функции рисования

Чтобы воссоздать эффект вращения треугольника, необходимо выполнить следующие два повторяющихся шага: (1) изменить текущий угол поворота треугольника и (2) вызвать функцию рисования, передав ей угол поворота. Далее нужно потребовать от браузера, чтобы он циклически выполнял эти действия. Для этого можно использовать функцию `requestAnimationFrame()`, которая 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.

Сгруппируем эти действия в одну анонимную функцию. Анонимная функция используется чтобы обеспечить передачу локальных переменных, объявленных в `main()` (`gl`, `n`, `currentAngle`, `modelMatrix` и `u_ModelMatrix`) функции `draw()`.

Изменение угла поворота

Алгоритм изменения текущего угла поворота поясняется рис. 7.

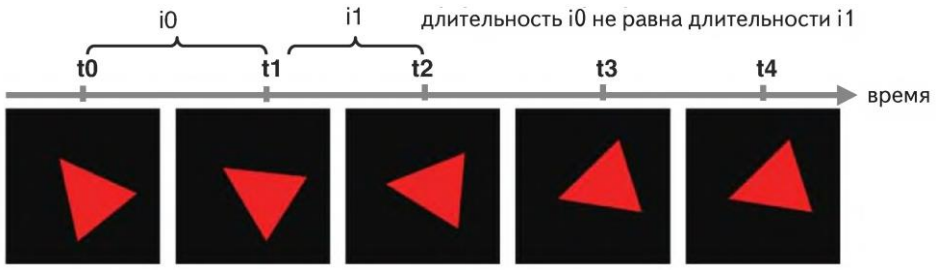

Рис. 7. Интервалы времени между вызовами `tick()` могут различаться

Интервалы между моментами времени t0 и t1, t1 и t2, t2 и t3 и т.д., могут быть разными, потому что нагрузка на браузер с течением времени может изменяться. То есть, разность t1-t0 может отличаться от разности t2-t1.

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 треугольника.

Поэтому создадим функцию, которая будет определять новый угол поворота, исходя из интервала времени, прошедшего с момента последнего ее вызова. Для этого создадим в ней 2 переменные:

1) переменную `g_last`, которая будет хранить время последнего вызова
2) переменную `now`, которая запоминает текущее время.

Затем в этой функции вычисляется продолжительность интервала времени, прошедшего с момента предыдущего вызова, с помощью разности `now-g_last`, и сохраняется результат в переменной `elapsed`. После этого, исходя из значения `elapsed`, вычисляется величина угла поворота:

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

Обеим переменным, `g_last` и `now`, присваивается значение, возвращаемое методом `now()` объекта `Date`, измеряемое в миллисекундах (1/1000 секунды). Поэтому, чтобы вычислить угол поворота, исходя из скорости вращения (градусы в секунду), достаточно просто умножить 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 вернуть вызывающей программе.