Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Институт повышения квалификации
и переподготовки

Кафедра «Профессиональная переподготовка»

В. А. Бельский

АНИМАЦИОННАЯ ГРАФИКА

ПРАКТИКУМ
по одноименному курсу
для слушателей специальности переподготовки
1-40 01 74 «Web-дизайн и компьютерная графика»
заочной формы обучения

Гомель 2019

УДК 004.928(075.8)
ББК 32.973я73
Б44

Рекомендовано кафедрой «Профессиональная переподготовка»
Института повышения квалификации
и переподготовки ГГТУ им. П. О. Сухого
(*протокол № 1 от 24.09.2018 г.*)

Рецензент: профессор кафедры «Информатика» ГГТУ им. П. О. Сухого
д-р физ.-мат. наук *В. П. Кудин*

Бельский, В. А.

Б44 Анимационная графика : практикум по одноим. курсу для слушателей специальности переподготовки 1-40 01 74 «Web-дизайн и компьютерная графика» заоч. формы обучения / В. А. Бельский. – Гомель : ГГТУ им. П. О. Сухого, 2019. – 57 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://elib.gstu.by. – Загл. с титул. экрана.

Содержится описание основных возможностей, которые предоставляет CSS3 для создания анимации. Для любителей языка JavaScript приведены многочисленные примеры создания анимации с использованием API Canvas, а также с применением библиотеки JQuery. Рассмотрены примеры создания gif-анимации с использованием программы Adobe Photoshop. В заключительной части пособия рассматривается процесс создания мультфильма с использованием программы Anime Studio.

УДК 004.928(075.8)
ББК 32.973я73

СОДЕРЖАНИЕ

Лабораторная работа №1. Анимация средствами CSS 4
Лабораторная работа №2. Анимация в Photoshop 21
Лабораторная работа №3. Анимация с использованием API Canvas.....28
Лабораторная работа №4. Анимация с использованием JQuery 43
Лабораторная работа №5. Создание мультфильма в Anime Studio.....53

Лабораторная работа №1. Анимация средствами CSS

Цель работы: с помощью различных команд CSS научиться реализовывать несложную анимацию на веб-странице.

Теоретические сведения и примеры кода

1) Свойство *transform*

Поворот (rotate)

transform: rotate(25deg); поворот на 25 градусов по часовой стрелке;

transform: rotate(-75deg); поворот на 75 градусов против часовой стрелки;

Масштабирование (scale)

transform: scale(2) – увеличение по x и по y в два раза;

transform: scale(2, 0.5) – увеличение в 2 раза по *x* и уменьшение в 2 раза по *y*;

transform: scaleX(3) – увеличение в 3 раза по *x*;

Рисунок 1.1 – Масштабирование элементов веб-страницы

transform: scale(-1) – поворот вокруг обеих осей;

transform: scale(1, -1) – поворот вокруг поворот вокруг оси *x*;

transform: scale(-1) – поворот вокруг обеих осей;

#twit3:active {transform: scale(-1, 1);} – CSS-код для элемента twit3 (на скриншоте видно, что второй элемент, при применении данного кода зеркально отразился справа налево; поворот осуществлен вокруг оси y).

Перемещение (translate)

transform: translate (20px, -10px) – перемещение вправо на 20 px и вверх на 10 px;

transform: translate (2px, 2px) – перемещение вправо на 2 px и вниз на 2 px;

transform: translateY (-30px) – перемещение вверх на 30 px;

transform: translateX (0.5em) – перемещение вправо на половину высоты шрифта;

```
#but:active {
    transform: translate(1px, 2px);
}
```

Приведенный выше CSS-код создает эффект нажатия на кнопку – при этом она смещается на немного вправо и вниз.

Наклон (skew)

```
#tube1:active {
    transform: skew(-45deg, 0);
}
```

Приведенный выше код наклоняет элемент #tube1 на 45 градусов по часовой стрелке. При этом наклон осуществляется вдоль горизонтальной оси (оси x).

transform: skew (45geg, 0) – на 45 против часовой стрелки;

transform: skew (0, -45deg) – на 45 против часовой стрелки, но на этот раз наклон происходит вдоль вертикальной оси (оси y);

transform: skew (-30geg, -50deg) – наклон по двум осям;

Приведенные выше виды трансформации можно применять одновременно.

Комбинированный эффект

Эффекты перечисляются после *transform* в произвольном порядке.

```
#witch4:active {
transform: scale(1.8) skew(-15deg, 0) rotate(45deg) translate(-150px);
}
```

В приведенном выше примере CSS-кода элемент с id witch4 последовательно подвергается следующим изменениям: увеличивается в 1.8 раз, наклоняется вправо на 15 градусов, поворачивается вправо на 45 градусов и, наконец, смещается влево и вверх на 45px.

Свойство transform-origin

Это свойство позволяет изменить точку трансформации объекта, т.е. точку, относительно которой преобразование будет осуществляться. По умолчанию, эта точка находится в центре элемента. До сих пор вся наша анимация осуществлялась относительно такой точки.

```
#tree2:active {
    transform-origin: 0% 100%;
    transform: rotate(45deg);
}
```

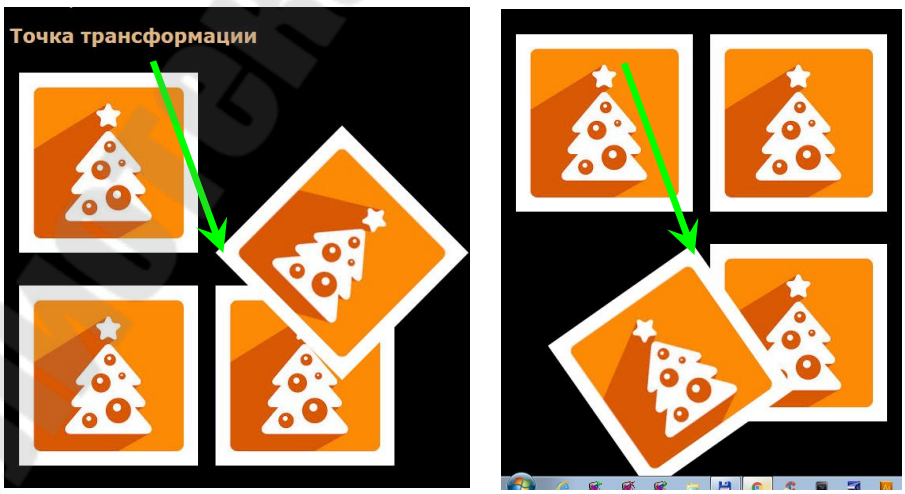

Рисунок 1.2 – Перенос точки трансформации объекта

Объясним этот код. Прежде, чем поворачивать вторую елочку на 45 градусов по часовой стрелке, мы сместили точку трансформации: она перешла в левый нижний угол рисунка (по x – 0%, т.е. перешла в левый край объекта, а по y – 100%, т.е. перешла в самый низ объекта).

```
#tree3:active {
    transform-origin: 100% 0%;
    transform: rotate(-35deg);
}
```

Точка трансформации объекта *#tree3* перешла в правый верхний угол.

Точка трансформации, разумеется, может находиться вообще за пределами элемента.

```
#tree4:active {
    transform-origin: -400px -400px;
    transform: rotate(-25deg);
}
```

2) Переходы *transition*

Переход является простой анимацией от одного набора CSS-свойств к другому через определенный промежуток времени.

Чтобы переход заработал, нужно следующее:

1) два стиля – начальный вид элемента и конечный вид;

2) свойство *transition.* Обычно применяется к исходному стилю;

3) инициатор – представляет собой действие, вызывающее изменение от одного стиля к другому. В CSS для запуска анимации можно использовать несколько псевдоклассов.

Можно анимировать:

- color, background-color;

- border-color, border-width;
- font-size, height, width;
- margin, padding;
- opacity;
- свойства позиционирования.

В основе CSS-переходов лежат четыре свойства, которые управляют тем,

какие свойства анимировать *transition-property;*

продолжительность анимации *transition-duration;*

тип анимации *transition-timing-function;*

задержка перед началом анимации *transition-delay.*

Синтаксис

Элемент{

свойство1: значение;

свойство2: значение;

свойство3: значение;

свойство4: значение;

transition-property: свойство2, свойство4;

}

Элемент:действие{

свойство2: новоеЗначениеСвойства2;

свойство4: новоеЗначениеСвойства4;

}

Анимация цвета – простой переход

.b1{

background-color: white;

```
color: darkblue;
transition-property: background-color, color;
}
.b1:hover{
background-color: darkblue;
color: white;
}
```

В рассмотренном выше примере анимируется элемент, к которому прикреплен CSS-класс .b1 (кнопка, в данном случае). Отметим, что анимируются два свойства: цвет фона и цвет текста. При наведении указателя на кнопку, цвет фона меняется на цвет текста, а цвет текста, в свою очередь, на цвет фона. Анимация происходит сразу же при наведении.

Анимация цвета – добавлено время анимации

```
.b2{
    background-color: white;
    color: darkblue;
    transition-property: background-color, color;
    transition-duration: 3s;
}
.b2:hover{
    background-color: darkblue;
    color: white;
}
```

Рисунок 1.3 – Анимация цвета

Длительность анимации задается свойством *transition-duration.*

Теперь анимация происходит не мгновенно, а длится в течение трех секунд.

В примере, предложенном ниже, анимируются несколько свойств.

Анимирование трех свойств: color, background-color, opacity

```
.b3{
background-color: white;
color: darkblue;
opacity: 1;
transition-property: background-color, color, opacity;
transition-duration: 3s;
}
.b3:hover{
    background-color: darkblue;
    color: white;
    opacity: 0;
}
```

Этот пример отличается от предыдущего тем, что, по происшествии трех секунд, свойство *opacity* кнопки становится равным нулю, а значит, кнопка становится невидимой.

Управление скоростью анимации

Скорость выполнения анимации контролируется с помощью свойства transition-timing-function, которое может принимать значения:

- linear;
- ease (по умолчанию);
- ease-in (медленнее в начале анимации);
- ease-out (медленнее в конце анимации);
- ease-in-out;

- cubic-bezier.

В предложенном ниже примере имеются четыре объекта – четыре рыбки. При наведении курсора мыши на объект, он движется вправо. Какое свойство следует анимировать, чтобы рыбка двигалась вправо? Здесь возможно не одно решение.

```
#fish1{
    position: relative;
    left: 0px;
    transition-property: left;
    transition-duration: 3s;
}

#fish1:hover{
    left: 600px;
}
```

Как видно из этого примера, мы анимировали CSS-свойство *left.*

```
#fish4{
    position: relative;
    left: 0px;
    transition-property: left;
    transition-duration: 3s;
    transition-timing-function: cubic-bezier(0.8, 0.1, 0.9, 0.1);
}
#fish4:hover{
    left: 1000px;}
```

А в этом примере использована еще одна функция скорости – *cubic-bezier*, дающая больший контроль над скоростью анимации.

Иногда необходимо, что бы элемент сохранил тот вид, который был у него по завершении анимации (например, мы хотим, чтобы рыбка осталась справа). Следующая опция заставит браузер сделать это.

animation-fill-mode: forwards

Наконец, ниже представлен еще один вариант того, как можно заставить объект двигаться вправо. На этот раз анимируется CSS-свойство *margin-left.*

```
#fish44{
    position: relative;
    /*margin-left: 10px;*/
    transition-property: margin-left;
    transition-duration: 3s;
}
#fish44:hover{margin-left: 1000px;}
```

3) Работа с ключевыми кадрами **(***keyframes***)**

С помощью CSS-переходов можно анимировать *только переход от одного набора CSS-свойств к другому*.

Анимация, предусмотренная в CSS3, позволяет анимировать переходы от *одного набора свойств к другому, затем к третьему…*

Можно задать повторяющуюся анимацию, приостановить ее выполнение при проходе указателя мыши над элементом и даже повернуть ее вспять, когда анимация дойдет до конца.

Анимация с помощью ключевых кадров в CSS3, сложнее перехода, но для нее *не требуется специальный инициализатор* (например, ***hover***).

Создание анимации с помощью *keyframes*

При создании анимации первым делом создаются ключевые кадры *(keyframes)*.

Создание анимации происходит в два приема:

1) Определение анимации

```
@keyframes имяАнимации{
    from{список CSS-свойств;}
    to{список CSS-свойств;}
}
```

2) Применение анимации к элементу

```
<div id = “element”></div>
#element:hover {
    animation-name: имяАнимации;
    animation-duration: 4s;
}
```

Рассмотрим несколько примеров.

Анимация цвета

```
@keyframes backgroundGlow{
    from{background-color: black;}
    to{background-color: yellow;}
}
body{
    animation-name: backgroundGlow;
    animation-duration: 1s;
}
```

В первом блоке приведенного выше кода определяется анимация с именем *backgroundGlow*. Анимация призвана изменять цвет фона с черного на желтый.

Во втором блоке эта анимация назначается элементу *body*. Анимация произойдет при загрузке страницы и будет длиться одну секунду.

Анимация цвета с промежуточными кадрами

Можно создавать более двух ключевых кадров, что гораздо более интересно. По такому принципу можно добавлять сколько угодно ключевых кадров – просто используйте необходимое число в процентах (30%, 60%, 75% и так далее).

```
@keyframes color1{
    from{background-color: red;}
    20%{background-color: black;}
    40%{background-color: red;}
    60%{background-color: black;}
    80%{background-color: red;}
    to{background-color: black;}
}
```

Эта анимация заставит цвет фона элемента, к которому она применена, изменяться несколько раз с красного на черный, что создает эффект мигания.

Отметим, что ключевые слова **from** и **to** равнозначны процентным записям **0%** и **100%** соответственно.

Анимация цвета с промежуточными кадрами. Анимация нескольких свойств

```
@keyframes color2{
from{background-color: red;}
50%{background-color: white;
    transform-origin: 0% 100%;
        transform: scale(1.5)}
to{background-color: black;
    transform-origin: 0% 100%;
        transform: scale(2.2)}
}
```

```
@keyframes color3{
from{background-color: red;}
25%{background-color: white;
        transform: scale(1.25);
        transform: rotate(720deg)}
75%{background-color: white;
        transform: scale(1.25);
        transform: rotate(720deg)}
to{background-color: black;
        transform: translate(-200px, -200px)}
}
```

Эти три вида анимации применяются к ячейкам таблицы 4x4.

```
#cell1:hover {
animation-name: color1;
animation-duration: 2s;}

#cell2:hover {
animation-name: color1;
animation-duration: 0.5s;}

#cell3:hover {
animation-name: color2;
animation-duration: 2s;}

#cell4:hover {
animation-name: color3; animation-duration: 4s;}
```

Рисунок 1.4 – Масштабирование с помощью ключевых кадров

Применение нескольких анимаций

Определим две анимации:

```
@keyframes smile1{
from{;}
to{transform: rotate(1080deg);}
}
@keyframes smile4{
from{opacity: 1;}
to{opacity: 0;}
}
```

Следующий код применяет эти анимации к одному объекту:

```
#s3:hover {
animation-name: smile1, smile4;
animation-duration: 2s, 2s;
}
```

Управление скоростью анимации

Свойство *animation-duration* позволяет указать продолжительность анимации.

Можно также указывать функцию распределения скорости анимации во времени *animation-timing-function***,** которая может принимать значения

- linear
- ease (по умолчанию)
- ease-in (медленнее в начале анимации)
- ease-out (медленнее в конце анимации)
- ease-in-out

Ниже мы приводим пример, в котором используются эти функции, а также функция cubic-bezier.

```
@keyframes smile3{
from{;}
to{transform: translateX(400px);} }
#cell_a:hover {
animation-name: smile3;
animation-duration: 1.5s;
animation-timing-function: ease-in;}
#cell_b:hover {
animation-name: smile3;
animation-duration: 1.5s;
animation-timing-function: ease-in-out;}
#cell_c:hover {
animation-name: smile3;
animation-duration: 1.5s;
animation-timing-function: linear;}
#cell_d:hover {
animation-name: smile3;
animation-duration: 1.5s;
animation-timing-function: cubic-bezier(1, 0.1, 1, 0.1);}
```

Повторение анимации

С помощью CSS-переходов можно контролировать еще несколько аспектов анимации, включая повторение и направление выполнения.

animation-iteration-count: 5; – повторить анимацию 5 раз;

animation-iteration-count: infinite; – повторить анимацию все время;

animation-delay: 2s; – задержка перед началом выполнения анимации 2 секунды;

animation-direction: alternate; – проиграть анимацию с конца, т.е. анимация проиграется из A в B (как было бы и без этой опции), а затем из B в A.

animation-direction: reverse; – анимация будет проигрываться сразу с конца в начало, т.е. из B в A.

Пример. Пульсирующие кнопки

Создаем две анимации, которые будучи примененными, изменяют масштаб объекта.

```
@keyframes pulse{
from{;}
20%, 40%{transform: scale(1.2);}
25%, 45%{transform: scale(1.0);}
to{;}
}
@keyframes pulse2{
from{;}
10%, 30%, 50%, 70%, 90%{transform: scale(1.05);}
20%, 40%, 60%, 80%, 100%{transform: scale(1.0);}
to{;}
}
```

Вот примеры использования этих видов анимаций. Создан эффект бьющегося сердца.

```
#hart1:hover {
animation-name: pulse;
animation-duration: 1.5s;
animation-delay: 2s;}
#hart2:hover {
```

```
animation-name: pulse;
animation-duration: 1s;
animation-delay: 1s;
animation-iteration-count: infinite;}
#butt:hover {
animation-name: pulse2;
animation-duration: 0.5s;
animation-iteration-count: 10;}
#but:hover {
animation-name: smile4;
animation-duration: 0.5s;
animation-iteration-count: 8;
animation-direction: alternate;}
```

Приостановка анимации

В приведенном ниже примере, елочка вращается бесконечно, но вращение останавливается при наведении мыши на нее.

```
@keyframes rot{
from{opacity: 1;}
to{transform: rotate(360deg);
                opacity: 0;} }
#tree {
animation-name: rot;
animation-duration: 0.7s;
animation-iteration-count: infinite;
animation-timing-function: linear;}
```

#tree:hover {

animation-play-state: paused;}

Задание к лабораторной работе №1

Используя изложенный выше теоретический материал, а также рассмотренные примеры,

1) создать несколько тематически связанных веб-страниц;

2) продемонстрировать работу со свойствами *transform* и *transition*;

3) выполнить анимацию с использованием ключевых кадров.

Контрольные вопросы

1. Приведите примеры использования свойства *transform*.

2. Может ли значение масштаба быть отрицательным?

3. Какие свойства могут анимироваться при использовании *transition*?

4. Что такое точка трансформации объекта и как получить к ней доступ?

5. Как регулировать продолжительность анимации?

6. Как изменить скорость анимации?

7. Как реализовать проигрывание анимации с конца в начало?

8. Как определить анимацию при помощи ключевых кадров?

9. Сколько промежуточных кадров может содержать анимация?

10. Как заставить анимацию выполниться пять раз?

11. Как зациклить анимацию?

12. Как остановить анимацию?

13. Как заставить браузер оставить объект в том положении, в котором он находился после завершения анимации?

Лабораторная работа №2. Анимация в Photoshop

Цель работы: научиться реализовывать вращающийся логотип и анимированный баннер в программе Photoshop.

1. Простое движение

В этом примере продемонстрировано простое движение, реализованное с использованием временной шкалы.

1) Ctrl + N – новый файл.
2) Теперь нужен какой-нибудь объект. Можно выбрать из панели инструментов «произвольная фигура». Потом нажмите растрировать на панели слоев.
3) Команда Окно – Анимировать
4) Выделяем кадр. Выставим время, например 0,1. Нажимаем создание копии выделенных кадров.
5) При нажатом втором кадре смещаем объект куда хотим.
6) Снова выделим первый кадр и нажмем создать промежуточные кадры. Введем число, например, 10. Появятся промежуточные кадры.
7) Все готово. Осталось Файл – Сохранить для Web и устройств.

2.

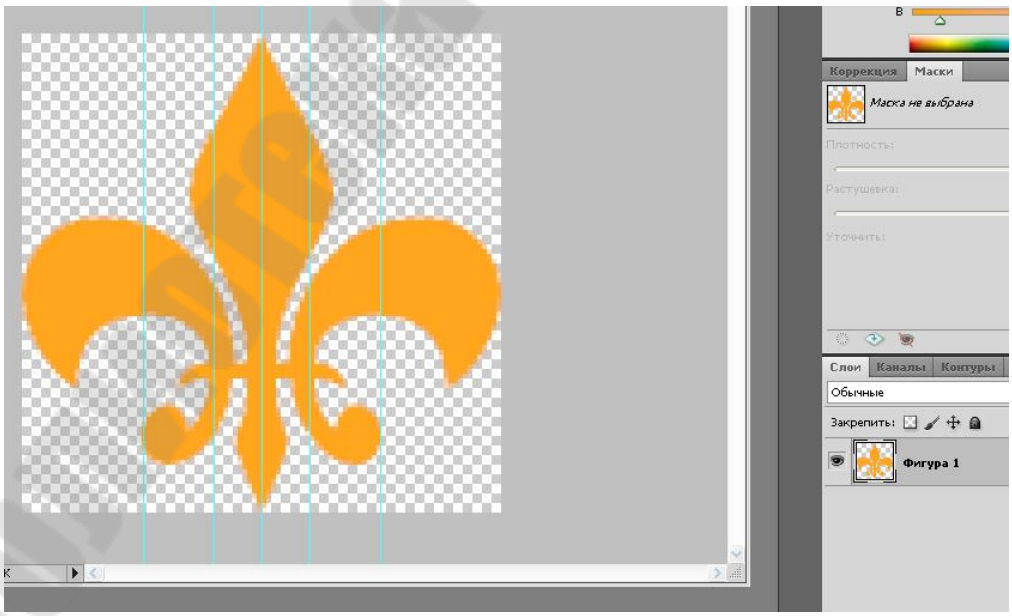

Рисунок 2.1 – Исходное положение объекта

Вращающийся элемент (например, логотип)

1) Ctrl + N – новый файл. Размеры, например, 100*100. Прозрачность выставить. Ctrl+ - увеличить область.

2) Добавим какую-нибудь фигурку. Можно воспользоваться инструментом «Произвольная фигура». Ctrl + T – растянем на всю область.

3) Растрируем слой. Добавим несколько вертикальных направляющих.

4) Дублируем слой. Отключаем нижний слой. Выделяем новый слой. Ctrl +T – дотягиваем до первых направляющих слева и справа. Подтверждаем трансформацию.

5) Проводим итерацию со следующими направляющими (дублирование, трансформация…).

6) Еще раз – ставим фигуру почти на ребро.

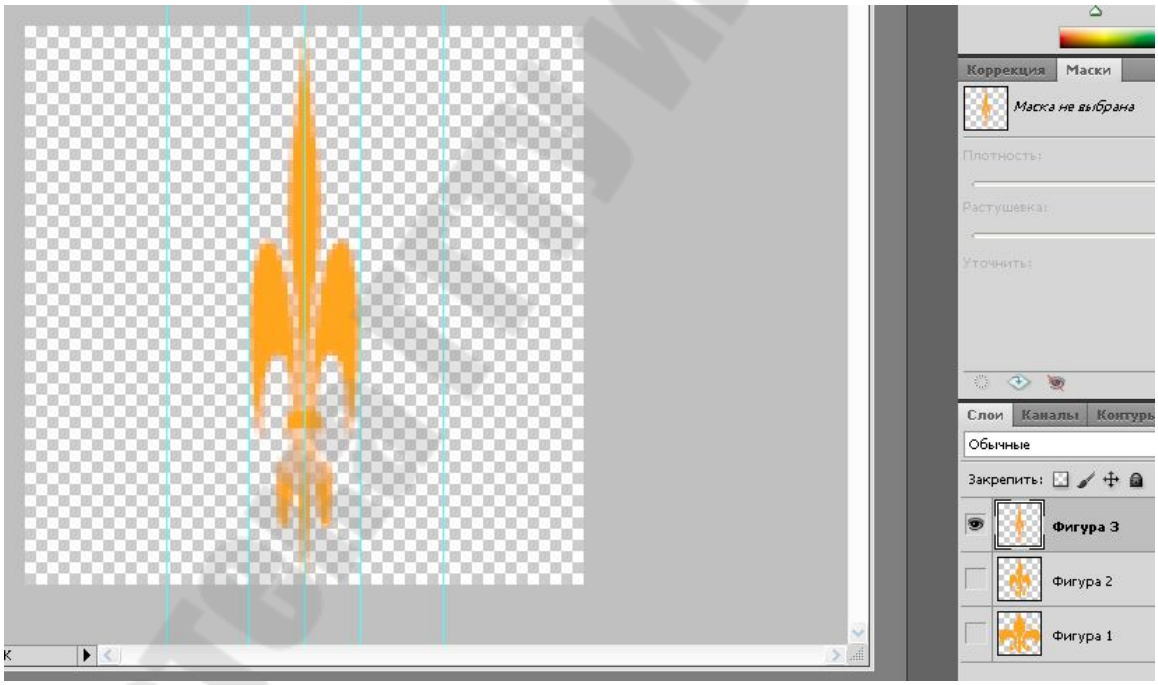

Рисунок 2.2 – Создаем слои

7) Приступаем к анимации. Окно-анимация. Появляется окно для покадровой анимации. Выделяем первый слой. Остальные отключаем. Этот слой появляется в первом кадре.

Выставим время 0,1.

8) В окне анимации нажмем создание копии выделенных кадров. И еще и еще – сделаем три копии. Теперь у нас четыре одинаковых кадра.

9) Нажмем на второй кадр – выключим все слои, кроме второго. По аналогии с остальными.

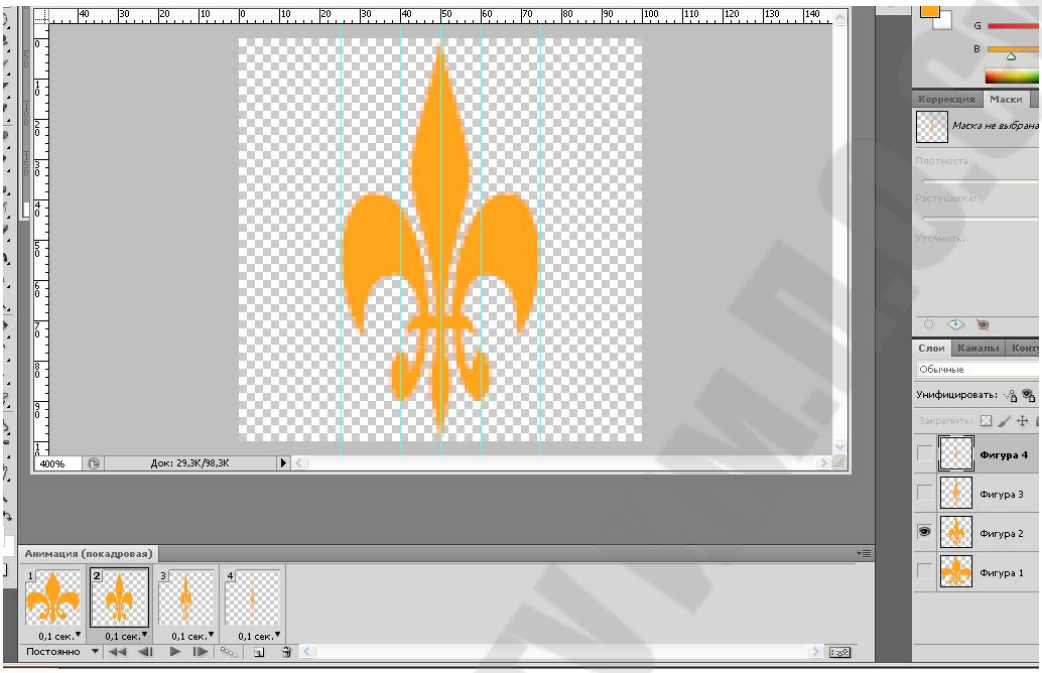

Рисунок 2.3 – Создаем кадры

10) Чтобы анимация не прыгала, повторим те же самые кадры в обратном порядке. На самом деле можно добавить только два промежуточных. Следить за тем, чтобы при создании кадра был выбран соответствующий слой! Получилась простая 6-кадровая анимация.

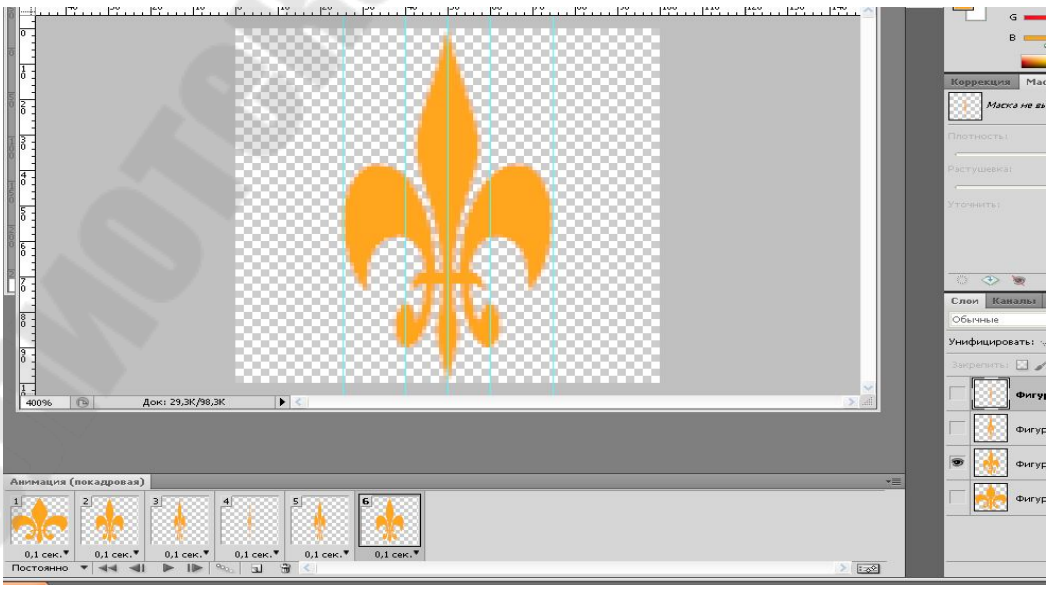

Рисунок 2.4 – Завершающие кадры

11) Можно сохранять. Файл – сохранить для веб и устройств. Анимация готова.

3. Анимационный баннер в фотошопе

Первый вариант

Анимационный баннер, полученный в результате простой смены картинок (фонов, рекламных объявлений). Для этого нам надо сделать следующие простые шаги:

1) Заготовить картинки для баннера, например 500*100.

2) Создать новый файл, выставить размер соответствующий размеру баннера.

3) Расположить эти картинки так, чтобы каждая заняла свой слой.

Рисунок 2.5 – Анимация смены слоев

4) Собственно анимация. Вызываем панель анимации. Сделаем так, чтобы нижний слой был в первом кадре (два других слоя следует отключить). Выставим время, например 2,0 с.

Второй слой во втором кадре (2 секунды). Аналогично третий.

5) Сохраняем для веб и устройств.

Второй вариант

Анимационный баннер, полученный в результате наложения на один и тот же фон-заготовку разных текстовых сообщений (или можно каких-то изображений):

3) Расположим картинку на первом слое. Создадим еще несколько слоев для текста (например, три): *text1, text2, text3*.

4) Напишем текст.

5) Создаем анимацию. Первый кадр – фон + text1. Остальные слои выключить. Аналогично второй и третий кадр. Время 2с или сколько нужно.

6) Сохранить.

Рисунок 2.6 – Сменяющийся текст на баннере

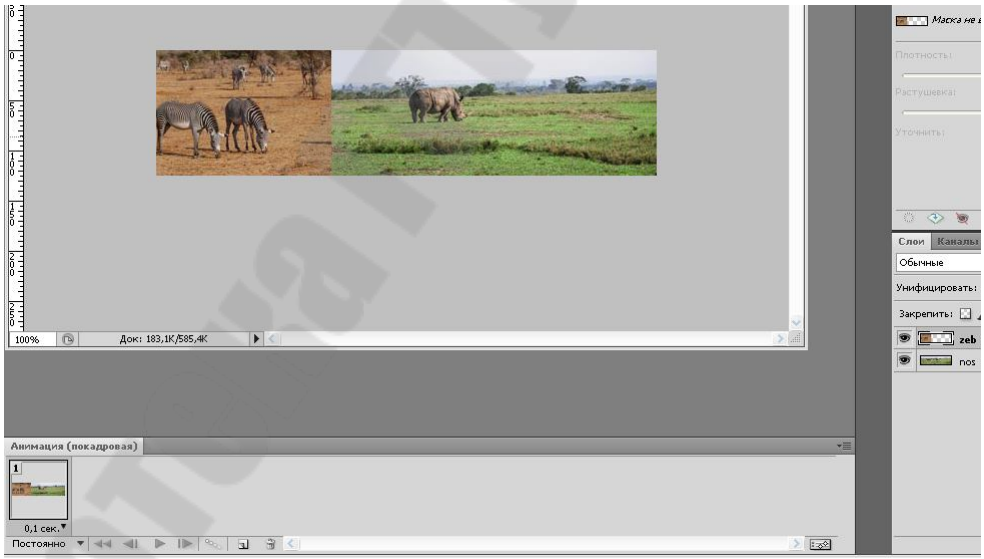

Рисунок 2.7 – Создаем сменяющиеся картинки

Третий вариант

Анимационный баннер, полученный в результате скользящих картинок):

1) Заготовить картинки для баннера, например 500*100.

2) Создать новый файл, выставить размер соответствующий размеру баннера. Для простоты возьмем две картинки.

3) Создайте два слоя – две картинки.

4) Зайдите в панель анимации. В первом кадре – оба слоя выделены. Выделите верхний слой и, нажав кнопку «перемещение» в панели инструментов смещайте картинку влево до тех пор, пока ее правый край не сравняется с левым краем нижней картинки.

Теперь у нас одна картинка не видна – находится слева, а вторая видна.

5) Время выставим 0,1. Нажмем «создание копии выделенных кадров». В этом кадре, манипулируя картинками и выделяя соответствующие слои, передвинем картинки вправо так, чтобы та картинка, которая была слева (за кадром) оказалась в кадре (стала видна), а та, которая была видна, ушла вправо.

6) Выделим первый кадр – создать промежуточные кадры – выберем 10. Проверьте, как картинки бегают.

7) После второй картинки анимация сразу перепрыгивает на первую картинку. Надо это исправить.

8) Последний кадр теперь имеет номер 12. Продублируем его. Время 12 кадра выставим в ноль. В 13-м кадре сместим картинку, которая сейчас находится справа за той, что видна, влево («пристыкуем»). Внешне 13 кадр не изменился – но картинка, которая была справа вне поля видимости, находится теперь слева (тоже вне поля видимости).

9) Продублируем 13 кадр. В 14-м кадре сместим все вправо.

10) Выделим 13-й кадр – создать промежуточные кадры – выберем 10.

11) В последнем (24-м) кадре лучше поставить время 0, т.к. он совпадает с первым кадром.

12) Сохранить.

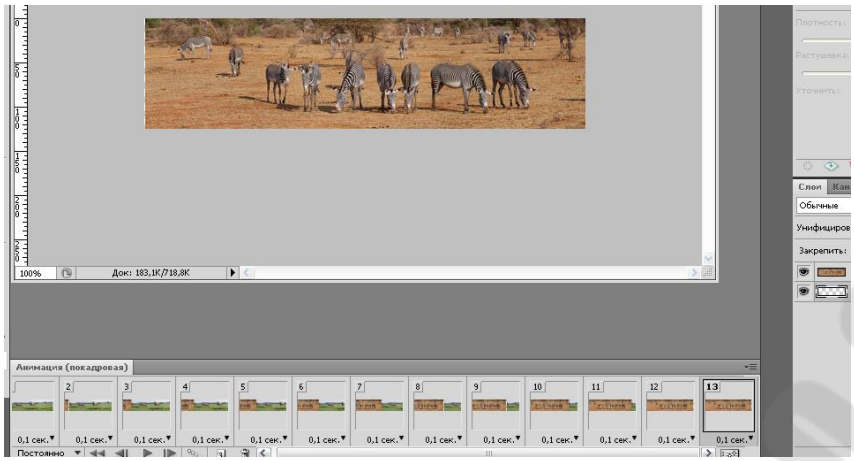

Рисунок 2.8 – Создаем промежуточные кадры

Задание к лабораторной работе №2

Используя изложенный выше теоретический материал, а также рассмотренные примеры,

1) создать произвольную веб-страницу;

2) создать на странице вращающийся элемент;

3) создать анимированный баннер.

Контрольные вопросы

1. Как получить доступ к временной шкале в программе Photoshop?

2. Как и с какой целью добавляются промежуточные кадры?

3. Опишите процесс создания вращающегося логотипа.

4. Опишите процесс создания слайдера.

5. Как экспортировать созданную анимацию в gif-формат?

Лабораторная работа №3. Анимация с использованием API Canvas

Цель работы: научиться работать с базовым интерфейсом API Canvas; научиться реализовывать анимацию графических элементов и текста на холсте.

Элемент *canvas*, добавленный в HTML5, предназначен для создания графики с помощью JavaScript. Например, его используют для рисования графиков, создания фотокомпозиций, анимаций и даже обработки и рендеринга видео в реальном времени.

1. Синтаксис. Основные команды

```
<canvas id="canvas"></canvas>


var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
ctx.fillStyle = "green";
ctx.fillRect(x, y, width, height);
ctx.arc(x, y, radius, startAngle, endEngle, clockwise);
ctx.fill();
ctx.beginPath();
ctx.strokeStyle = "green";
ctx.lineWidth = 5;
ctx.stroke();
```

Прокомментируем этот код.

1-я строка: создаем Canvas на HTML-странице;

2-я строка: сохраняем в переменной *canvas* ссылку на элемент Canvas;

3-я строка: получаем доступ к рисованию на холсте;

4-я строка: задаем цвет заливки;

5-я строка: определяем прямоугольник с координатами левого верхнего угла, а также шириной и высотой;

6-я строка: определяем дугу с началом в точках *x* и *y*, указываем какой угол начальный, какой конечный; последний параметр определяет направление рисования дуги – против часовой стрелки;

7-я строка: команда залить область;

8-я строка: начать путь (для рисования линией);

9-я строка: цвет линии;

10-я строка: толщина линии;

11-я строка: нарисовать линию.

2. Рисуем две пересекающиеся наклонные линии

```
ctx.beginPath();
ctx.lineWidth = 5;
ctx.moveTo(10, 10); //передвижение без рисования линию до точки (10, 10)
ctx.lineTo(400, 400); //нарисовать линию до точки (400, 400)
ctx.moveTo(10, 400);
ctx.lineTo(400, 10);
ctx.stroke();
```

3. Рисуем оранжевый смайлик

```
ctx.strokeStyle = "orange";
ctx.lineWidth = 5;
ctx.beginPath();
ctx.arc(225, 225, 200, 0, 2*Math.PI);
ctx.moveTo(185, 190);
ctx.arc(155, 190, 30, 0, 2*Math.PI);
ctx.moveTo(325, 190);
ctx.arc(295, 190, 30, 0, 2*Math.PI);
```

Рисунок 3.1 – Рисование дугами

ctx.moveTo(365, 225);

ctx.arc(225, 225, 140, 0, Math.PI, false);

ctx.stroke();

4. Приступаем к движению

В примере, рассматриваемом ниже, квадрат движется, повинуясь нажатию клавиш со стрелками. Данный пример, кроме приема анимации, демонстрирует нам событийную модель JavaScript, а также показывает, как работать со «слушателем события».

Логика такова: функция *draw* – для отрисовки квадрата; функция *move* – уничтожает текущий квадрат путем очистки холста и рисует квадрат в новой позиции; функция *handler* – в случае, если произошло одно из четырех событий (четыре клавиши), выполняет функцию *move*; слушатель события прикреплен к элементу *body*. При нажатии на любую кнопку клавиатуры, *body* это слышит и выполняет функцию *handler*.

var draw = function(x, y) {

ctx.fillRect(x, y, 50, 50);

}

var move = function(speedX, speedY) {ctx.clearRect(0,0, can.width, can.height);

initX += speedX;

initY += speedY;

draw(initX, initY);

}

Прикрепляем слушатель.

body.addEventListener("keydown", handler);

function handler(event) {

if(event.keyCode == 37) {move(-speed, 0);}

if(event.keyCode == 38) {move(0, -speed);}

```
    if(event.keyCode == 39) {move(speed, 0);}
    if(event.keyCode == 40) {move(0, speed);}
}
```

(37, 38, 39, 40 – коды клавиш со стрелками, соответственно, «влево», «вверх», «вправо», «вниз»).

5. Простая анимация

Сделать анимацию из элемента на холсте HTML5 достаточно просто. Для этого устанавливается таймер, который постоянно вызывает элемент, обычно 30 или 40 раз в секунду. При каждом вызове код полностью обновляет содержимое всего холста. Если код написан правильно, постоянно сменяющиеся кадры сольются в плавную, реалистичную анимацию.

JavaScript предоставляет два способа для управления этим повторяющимся обновлением содержимого холста:

1) ***Функция setTimeout()***

Эта функция дает указание браузеру подождать несколько миллисекунд, а потом исполнить фрагмент кода, в данном случае код для обновления содержимого холста.

Синтаксис

var timerId = setTimeout(func / code, delay);

Отмена исполнения – *clearTimeout().*

Функция setTimeout возвращает числовой идентификатор таймера timerId, который можно использовать для отмены действия.

Синтаксис

var timerId = setTimeout(func / code, delay);

clearTimeout(timerId);

2) ***Функция setInterval()***

Метод setInterval имеет синтаксис, аналогичный setTimeout.

var timerId = setInterval(func / code, delay);

Смысл аргументов – тот же самый. Но, в отличие от setTimeout, он запускает выполнение функции не один раз, а регулярно повторяет её через указанный интервал времени. Остановить исполнение можно вызовом *clearInterval(timerId)*.

Пример 1. Создание прямоугольников, рождающихся сверху холста в произвольной позиции и падающих вниз с произвольной скоростью

Рисунок 3.2 – Падающие вниз квадраты

1) Определяем класс Rect (можно было бы и не создавать класс, но поскольку фигур у нас будет много, то создание шаблона – хорошее решение).

```
var Rect = function() {
    this.x = Math.round(Math.random()*(can.width-size));
    this.y = -Math.round(Math.random()*20);//квадраты должны появ//ляться выше холста – этим объясняется знак «минус»
    this.speed = Math.floor(Math.random()*10) + 1;
};
```

Вновь созданный объект класса Rect будет иметь случайную координату по x в пределах ширины холста. Начальная координата по y будет такой, что бы объект сначала был не виден (находится сверху холста). Скорость объекта задается в пределах от 1 до 11.

2) Функция draw будет заливать прямоугольник случайным цветом.

```
Rect.prototype.draw = function() {
var color1 = Math.floor(Math.random()*255);
var color2 = Math.floor(Math.random()*255);
var color3 = Math.floor(Math.random()*255);

ctx.fillStyle = "rgb(" + color1 + "," + color2 + "," + color3 + ")";
ctx.fillRect(this.x, this.y, size, size);
};//end Rect.draw
```

3) Создание метода *move* у класса *Rect*

```
Rect.prototype.move = function() {
    ctx.clearRect(this.x, this.y, size, size);//перед тем как передвинуть //квадрат, нужно очистить место, которое он занимает в текущий момент
    this.y += this.speed;
};
```

4) Применение *setInterval*

```
setInterval(newRect, 10);
function newRect() {
    count++;
    info2.value = count;//счетчик созданных объектов
    var rect = new Rect();
    var index = setInterval(go, 50);
    function go() {
        rect.move();
        rect.draw();
    }
```

}

Отметим, что при такой реализации, все наши созданные объекты, даже те которые давно «упали» далеко ниже края холста, все равно «живут». В дальнейшем, мы будем уничтожать объекты, которые выходят за край холста.

Пример 2. Создание прямоугольников, рождающихся в центре холста и разлетающихся в стороны

Рисунок 3.3 – Квадраты, вылетающие из центра

1) Изменение в классе Rect

A. Теперь объекты создаются не сверху, а рождаются в центре холста

B. Теперь у них две скорости: по оси X и по оси Y

var Rect = function() {

this.x = (can.width-size)/2;

this.y = (can.height-size)/2;

*this.speedX = Math.floor(Math.random()*20) - 10;*

*this.speedY = Math.floor(Math.random()*20) - 10;*

};

2) Изменение в методе move

Rect.prototype.move = function() {

ctx.clearRect(this.x, this.y, size, size);

this.x += this.speedX;

this.y += this.speedY;

};

6. Работа с текстом в Canvas

Контекст рендеринга canvas предоставляет два метода для рисования текста:

fillText(text, x, y [, maxWidth])

– Вставляет заданный текст в положении (x, y). Опционально может быть указана максимальная ширина.

strokeText(text, x, y [, maxWidth])

– Вставляет контур заданного текста в положении (x, y). Опционально может быть указана максимальная ширина.

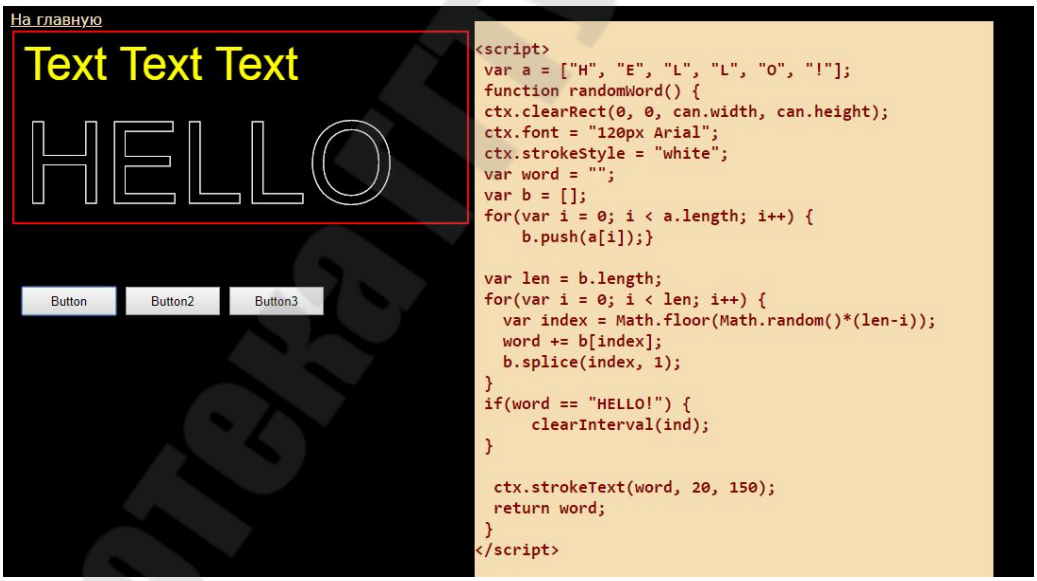

Рисунок 3.4 – Создание текста на холсте

Пример1. fillText, strokeText

ctx.font = "48px Arial";

ctx.fillText(“Text Text Text", 10, 50);

ctx.font = "120px Arial";

ctx.strokeStyle = "white";

ctx.strokeText("HELLO", 10, 200);

Пример 2. Движущийся текст. Реализация эффекта подбора пароля

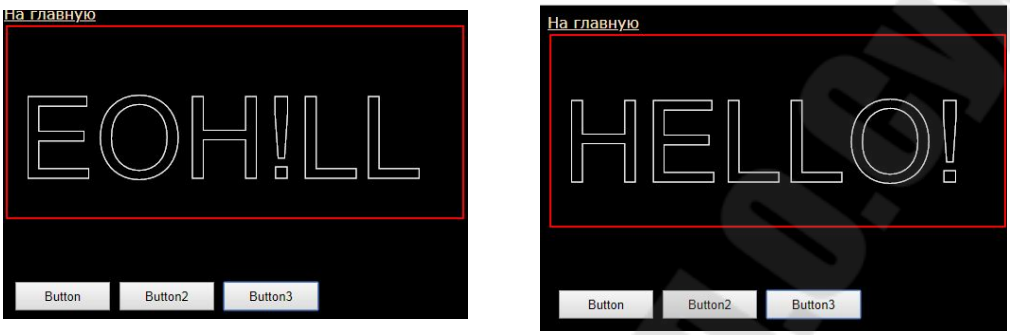

Рисунок 3.5 – Анимация текста

Из букв и восклицательного знака формируется массив. При каждом вызове функции, являющейся аргументом функции setInterval, на основе исходного массива, случайным образом формируется новый массив. Этот массив склеивается в слово. Как только это слово совпадет со словом «HELLO!», вызов функции прекращается. При этом в силу того, что каждый раз слово отрисовывается в случайной позиции, создается эффект дрожания или плавания слова.

Приводим код полностью, чтобы читатель мог отследить все нюансы.

```
window.onload = init;
function init() {
    var can = document.getElementById("canvas");
    var but1 = document.getElementById("button1");
    var but2 = document.getElementById("button2");
    var but3 = document.getElementById("button3");
    var ind;
    but1.addEventListener("click", clickHandler);
    but2.addEventListener("click", clickHandler2);
    but3.addEventListener("click", clickHandler3);
```

(Создали холст, кнопки и навесили на них слушатели)

```
can.width = 480;
can.height = 200;
var ctx = can.getContext("2d");
ctx.fillStyle = "yellow";
ctx.font = "48px Arial";
ctx.fillText("Text Text Text", 10, 50);
```

(Параметры текста)

```
function clickHandler() {
ctx.font = "120px Arial";
ctx.strokeStyle = "white";
ctx.strokeText("HELLO", 10, 180);
}
```

(Нажатие на первую кнопку)

```
var a = ["H", "E", "L", "L", "O", "!"];
function randomWord() {
ctx.clearRect(0, 0, can.width, can.height);
ctx.font = "120px Arial";
ctx.strokeStyle = "white";
var word = "";
var b = [];
for(var i = 0; i < a.length; i++) {
    b.push(a[i]);
}
var len = b.length;
for(var i = 0; i < len; i++) {
```

```
            var index = Math.floor(Math.random()*(len-i));
            word += b[index];
            b.splice(index, 1);
        }
        if(word == "HELLO!") {
            clearInterval(ind);
        }
        ctx.strokeText(word, 20, 150);
        return word;
    } //конец функции randomWord
```

Функция randomWord возвращает сформированное слово из букв, помещенных в массив *a*.

Функция randomWord2 отличается от первой добавленным эффектом дрожания слова.

```
    function randomWord2() {
        //здесь код повторяется
        var x = Math.round(Math.random()*20) - 10;
        var y = Math.round(Math.random()*20) - 10;
        ctx.strokeText(word, 20+x, 150+y);
        return word;
    }
```

Ниже выписан код функций, выполняющихся при нажатии на оставшиеся две кнопки.

```
        function clickHandler2() {
            ind = setInterval(randomWord, 10);
        }
        function clickHandler3() {
            ind = setInterval(randomWord2, 10);
```

```
    }
}// конец функции init
```

7. Работа с объектом Image в Canvas

В компьютерной графике проводится различие между векторной и растровой графикой. Естественно рисовать на холсте примитивами очень неудобно и требует определённых трудозатрат, и результат иногда явно хромает качеством. Поэтому естественно в canvas api предусмотрено взаимодействие с изображениями. Добавление изображения условно можно разделить на два шага: создание JavaScript объекта Image, а второй и заключительный шаг это отрисовка изображения на холсте при помощи функции drawImage. Рассмотрим оба шага подробнее.

1) Создание нового графического объекта:

```
var picture = new Image();
 picture.src = 'img.png';
```

2) Рисование изображения на холсте:

```
picture.onload = function() {
    ctx.drawImage(picture, x, y);
}
```

Пример 1. Смайлик, бегающий по картинке вправо влево

```
var img2 = new Image();
img2.src = "../img/smile.png";
img2.onload = function() {
setInterval(move, 20);
function move() {
    ctx2.clearRect(0, 0, w, h);
    if(flag) {
        ctx2.drawImage(img2, x+=5, 180);
```

```
            if(x > 500) flag = false;}
        if(!flag) {
            ctx2.drawImage(img2, x-=5, 180);
            if(x < 0) flag = true;}
    }};
```

Изменим функцию move, так чтобы смайлик, отталкивался не только от левого и правого края холста, но и от верхнего и нижнего. Сделаем так, чтобы при каждом столкновении с краем холста, скорость смайлика случайным образом изменялась.

```
function move() {
ctx2.clearRect(0, 0, w, h);
ctx2.drawImage(img2, x+=speedX, y+=speedY);
if(x >= 500 || x <= 0) {
    speedX = -speedX + Math.round(Math.random()*21);}
if(y >= 380 || y <= 0) {
    speedY =-speedY + Math.round(Math.random()*2-1);}
}
```

Рисунок 3.6 – Смайлик движущийся поверх картинки

Функция *.drawImage* с девятью параметрами

```
drawImage(img, x1, y1, width1, height1, x2, y2, width2, height2);
img.onload = function() {
    ctx.drawImage(img, 10, 10);
    ctx2.drawImage(img, 250, 350, 300, 300, 30, 30, 100, 100);
};
```

Эта функция копирует из картинки img нужный кусочек с параметрами *x1, y1, width1, height1* и помещает его в новую позицию *x2, y2*, производя при этом масштабирование.

Задание к лабораторной работе №3

Используя изложенный выше теоретический материал, а также рассмотренные примеры,

1) создать произвольную веб-страницу;

2) используя интерфейс API Canvas создать статический рисунок из графических примитивов;

3) создать анимацию на холсте, используя функции setTimeOut и setInterval;

4) показать работу с текстом на холсте.

Контрольные вопросы

1. Как создать холст?

2. Как выставить нужные размеры холста?

3. Как нарисовать дугу на холсте?

4. Применение функции setTimeOut для создания анимации на холсте.

5. Применение функции setInterval для создания анимации на холсте.

6. Что такое слушатель события?

7. Как остановить анимация, созданную с помощью функции set-Interval?

8. Как создать текст на холсте?

9. Как отформатировать текст на холсте?

10. Как поместить на холст изображение из графического файла?

Лабораторная работа №4. Анимация с использованием JQuery

Цель работы: научиться основам работы с библиотекой JQuery; научиться реализовывать анимацию графических и текстовых элементов веб-страницы с помощью JQuery.

Библиотека jQuery содержит несколько кросс-браузерных методов для анимации элементов, например, скольжение и плавное исчезновение, без привлечения дополнительных библиотек или плагинов. Для расширения возможностей работы с анимацией воспользуйтесь библиотекой jQuery UI (http://jqueryui.com), которая содержит набор интерфейсных взаимодействий, эффекты, виджеты и темы.

Подключение библиотеки производится очень просто.

```
<script src="../jquery-3.1.0.min.js"></script>
```

Используя данную библиотеку, очень удобно обращаться к элементам страницы:

```
<script>
$(“button”)
$(“h1”)
$(“#but”)
$(“.big”)
$(“div.myClass”)
$(“p#p21”)
</script>
```

Обратившись к нужному элементу, легко заставить выполнить его необходимую команду:

```
<script>
$(“button”).hide();
$(“h1”).toggle();
$(“#but”).show();
$(“.big”).slideUp();
```

$(“div.myClass”).slideDown();

$(“p#p21”).fadeOut();

</script>

1. Анимационные эффекты JQuery

Метод *.fadeIn()*

Управляет прозрачностью, показывая скрытый элемент, при этом свойство *opacity* выбранного элемента изменяется от 0 до 1. Для этого на странице появляется необходимое пространство для элемента, при этом остальные элементы могут сдвинуться с места.

.fadeIn(длительность, функция по завершении анимации)

– **длительность** – необязательный параметр, задает скорость проявления эффекта с помощью ключевых слов "fast", "normal", "slow" или числовых значений. По умолчанию используется значение "normal", равное 400 миллисекундам.

– **функция по завершении анимации** — необязательный параметр, задает функцию, которая будет вызвана после проявления элемента.

```
$("document").ready(init());
        function init() {
        $("p").hide();
        $("p").fadeIn(10000);
}
```

Метод *.fadeOut()*

Заставляет элемент исчезнуть, сделав его прозрачным, при этом остальные элементы могут сдвинуться с места. Свойство *opacity* выбранного элемента изменяется от 1 до 0.

.fadeOut(длительность, функция по завершении анимации)

```
$("document").ready(init());
```

```
function init() {
        $("p").hide();
        $("p").fadeIn(10000);
        $("button").click(function() {
                $("h1").fadeOut();
                $("img").fadeOut(5000);
        });
}
```

В приведенном выше примере при загрузке HTML-страницы происходят следующие события:

1) скрывается абзац;

2) абзац снова появляется на странице в течение 10 секунд;

3) при нажатии на кнопку, заголовок исчезает в течение 400мс, а картинка исчезает в течение 5с.

Метод *.show()*

Показывает ранее скрытый элемент. Если не задано значение скорости, то элемент появляется моментально, если скорость задана, то элемент появляется от верхнего левого к нижнему левому углу.

```
$("h1").show();
$("p").show(5000);
```

Метод *.toggle()*

Метод переключает выбранный элемент из одного состояния в другое, в зависимости от его текущего состояния, скрывая или отображая его.

```
$("h1").toggle();
```

Метод *.slideDown()*

Заставляет скрытый элемент появиться на веб-странице. Элемент проявляется постепенно – сначала его верхняя часть, и по мере проявления остальной части то, что находилось под элементом, сдвигается вниз. Поэтому для того, чтобы контент не перемещался по странице,

можно использовать абсолютное позиционирование элемента *{position:absolute;}*. Если необходимо поместить данный элемент относительно другого, то задайте относительное позиционирование {*position:relative;*} для элемента, который окружает абсолютно позиционированный элемент.

Метод *.slideUp()*

Изменяет свойство *height* элемента, пока она не станет равной 0, после скрывает элемент *display: none;*. При этом удаление начинается снизу, и если для элемента не задано позиционирование, то контент, находившийся ниже, перемещается вверх.

Метод *.slideToggle()*

Скрывает видимый элемент и показывает скрытый, т.е. может использоваться в качестве переключателя, позволяющего как отображать, так и скрывать элемент. При этом элемент проявляется постепенно, сверху вниз.

2. Пример с использованием функции setInterval()

```
$("document").ready(init());
function init() {
var index;
$("#button1").click(function() {
        $("h1").fadeOut("fast");
        $("h1").fadeIn("slow");
        $("img").show();
        index = setInterval(function() {
              $("p").hide("slow");
              $("p").show("fast");

        }, 2000);
});
```

```
$("#button2").click(function() {
        clearInterval(index);
        $("img").hide(4000);
});
}
```

Прокомментируем приведенный выше код. Нажатие на кнопку *button1* вызывает следующие действия:

1) выцветание и появление заголовка *h1*;

2) появление картинки;

3) инициирует функцию, которая попеременно скрывает и показывает абзац, эта функция вызывается каждые 2с; отметим, что для корректной работы этой функции, суммарное время выполнения функций *hide* и *show* не должно превышать двух секунд.

Нажатие на кнопку *button2* вызывает следующие действия:

1) используя переменную *index*, останавливает анимацию абзаца, запущенную нажатием первой кнопки;

2) прячет картинку.

3. Пример цепочки анимации

```
$("p").slideUp(2000).show(2000).fadeOut(2000).fadeIn(2000).hide(2000);
```

4. Создание собственных эффектов с помощью метода *.animate()*

Эффекты, которых нет в библиотеке jQuery, можно создавать с помощью метода *.animate()*. Интерпретатор браузера динамически, без перезагрузки страницы, изменяет выбранные свойства на указанные значения. Анимация происходит для всех элементов обернутого набора.

Чтобы добавить эффекты для конкретного элемента, нужно воспользоваться фильтрами jQuery для отбора.

Метод позволяет анимировать любое css-свойство, имеющее числовое значение, например, font-size, opacity, border-width, margin, padding, height, width, background-position и т.д. При этом имена свойств должны быть указаны слитно – fontSize, paddingLeft, или должен использоваться css-эквивалент свойства – "font-size". Числовые значения свойств не заключаются в кавычки.

Для любого свойства предварительно должно быть установлено начальное значение, а в css-объявлении должна использоваться полная запись каждого свойства, т.е., вместо свойства border должно быть заданы значения для border-style, border-width и т.д.

Значениями свойств могут также выступать hide, show или toggle, в результате чего к элементу применится вычисляемое значение – отображение, скрытие или переключение исходных состояний свойств.

Пример 1. Перемещение элемента страницы в заданное положение

```
$("div").animate({left: "200px", top: "200px"}, 500);
```

Данная анимация одновременно применяется к свойству left, для которого задано значение 200px, и к свойству top со значением 200px, продолжительность анимации задана 500ms.

Для элемента можно задавать относительное перемещение при каждом вызове анимации с помощью операторов +=, -=, *=, /=, например, $("div").animate({left: "+=200", top: "-=200"}, 500);

Пример 2. Смещение картинки в случайное положение

```
img {
    position: fixed;
    top: 450px;left: 10px;
}
$("#button2").click(function() {
    var left = Math.random()*800;
    var top = Math.random()*600;
```

```
$("img").animate({left:left+"px", top:top+"px"}, 1000);
});
```

Все анимируемые свойства должны задаваться в виде *отдельного числового значения*; нечисловые свойства не могут быть анимированы с использованием базового функционала jQuery. (Например, свойства width, height или left могут быть анимированы, но свойство background-color не может. Существуют отдельные плагины для анимации таких свойств, как color, background-color, border-color). Значения свойств без единиц измерения трактуются в виде количества пикселей. Единицы измерения em и % могут указываться, только в тех случаях, когда это применимо.

Сокращенные свойства CSS (например, margin, background, border) не поддерживаются. Например, если вам необходимо произвести анимацию свойства margin, то нужно указать каждое из отдельных свойств:

```
$(elem).animate({ marginTop: 10, marginRight: 20, marginBottom: 30, marginLeft: 40);
```

Пример 3. Анимирование свойств width, height и fontSize

Предположим, у нас есть абзац

```
p {
font-family: Arial, sans-serif;
font-size: 5px;
background-color: white;
color: red;
width: 100px;
height: 50px;
}
```

Анимация:

```
$("p").hover(function() {
$("p").animate({width: 500, height: 300, fontSize: 30}, 4000);
});
```

А этот блок кода отличается от предыдущего тем, что анимация абзаца вызывается наведением мыши на другой элемент (картинку):

$("img").hover(function() {

$("p").animate({width: 200, height: 100, fontSize: 5}, 4000);

});

5. Методы для CSS-стилей

.css()

.css(styleName):string

Возвращает значение css-величины styleName у выбранного элемента. Если выбрано несколько элементов, то значение будет взято у первого.

Пример 1. Получаем доступ к CSS-свойству *width* элемента p

$("#button1").click(function() {

var styleName = $("p").css("width");

document.getElementById("p1").innerHTML = styleName;//записываем в //другой абзац

});

Чтобы добавить какому-либо элементу стиль, необходимо воспользоваться следующим методом: *.css(name,value)*.

Пример 2.

$("div").css("border", "1px solid blue");

Данная инструкция обведет div синей рамкой.

Пример 3. При нажатии на кнопку назначается целый ряд CSS-свойств

$("#button1").click(function() {

$("p").css("border", "4px solid yellow");

$("p").css("box-shadow", "4px 4px red");

$("p").css("width", "600");

$("p").css("background-color", "#838949");

```
$("#button2").css("display", "none");
});
```

Если необходимо задать для элемента несколько CSS-правил, то лучше использовать следующую конструкцию:

```
.css({properties})
```

Пример 4.

```
$("div").css({ border:"1px solid blue", fontWeight:"bolder", backgroundColor:"red"});
```

Данная инструкция обведет div синей рамкой, сделает фон красным, а текст – жирным.

В примере 3 CSS-свойства добавлялись по одному. Можно добавить несколько свойств сразу:

```
$("img").hover(function() {
$("img").css({
    border: "8px solid blue",
    width: 200,
    boxShadow: "5px 5px green",
    left: 200
});
});
```

Задание к лабораторной работе №4

Используя изложенный выше теоретический материал, а также рассмотренные примеры,

1) создать произвольную веб-страницу или несколько тематически связанных страниц;

2) используя различные команды библиотеки JQuery создать различные анимационные эффекты на страницах.

Контрольные вопросы

1. Как подключить библиотеку JQuery?

2. Как, используя библиотеку, получить доступ к объекту страницы?

3. Основные анимационные эффекты JQuery.

4. Какие аргументы может принимать функция *fadeIn*?

5. Как создать цепочку анимаций?

6. Как создавать эффекты, которых нет в библиотеке JQuery?

7. Какие свойства позволяет анимировать метод *animate*?

8. Какие методы предусмотрены в библиотеке JQuery для работы с CSS-стилями?

Лабораторная работа №5. Создание мультфильма в Anime Studio

Цель работы: познакомиться с базовым интерфейсом программы Anime Studio; получить представление о различных видах анимации и приемах, применяемых при разработке анимации; получить практический навык поэтапной разработки мультипликационного фильма.

Anime Studio – программа для профессионалов. В этой среде можно разрабатывать мультфильмы любого уровня сложности и стиля. Anime Studio позволяет создавать различные виды анимации, в том числе, покадровую анимацию, морфинг, *tween*-анимацию, анимацию с использованием костей и другие виды.

Здесь мы остановимся только на некоторых особенностях работы в Anime Studio. Более подробные сведения содержатся в электронном конспекте лекций и в соответствующих руководствах.

Использование глубины слоев

Часто чтобы показать перспективу, мы используем глубину слоев. У каждого слоя есть z-индекс. По умолчанию он равен 0. Объект с z-индексом, равным 1, должен перекрывать объект с z-индексом, равным -2. Рассмотрим следующий пример.

Создадим слои с геометрическими объектами разных цветов. Поместим все слои в папку. Все слои получают z-индекс, равный 0. При этом, тот слой, который в папке лежит выше, перекрывает слои, лежащие под ним. Это обычное поведение слоев.

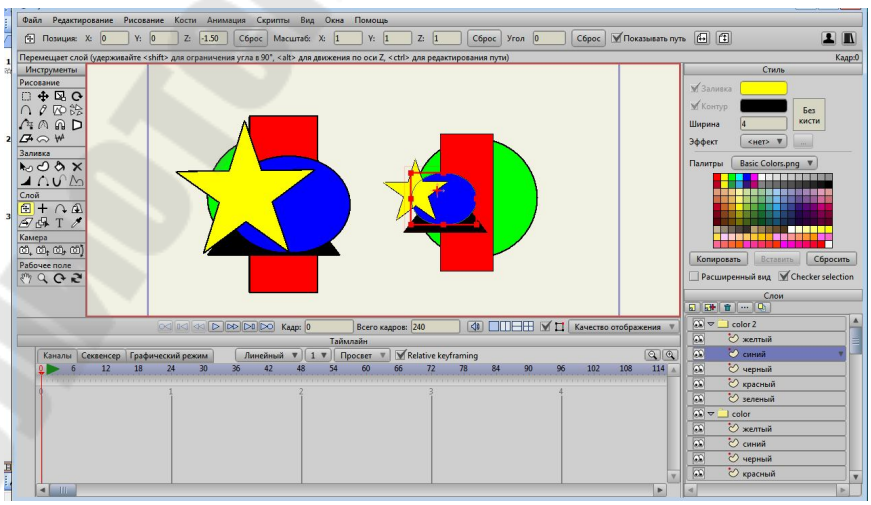

Рисунок 5.1 – Работа со слоями в Anime Studio

Продублируем всю папку, и теперь зададим z-индексы нашим слоям. Желтой звезде присвоим z: -2, синему эллипсу зададим z: -1.5 и так далее. Зеленому кругу присвоим z: 0. Заметим, что в соответствии с z-индексами размеры фигур изменились. Действительно, желтая звезда находится дальше всех и она самая маленькая, как и должно быть. Но она по-прежнему перекрывает другие объекты. Нас это не может устроить. Мы как раз хотим, чтобы объект с большим z-индексом перекрывал остальные (в нашем случае, это зеленый круг).

Продублируем еще раз всю папку. В третьей папке выставим опцию: *сортировать слои по глубине*. Теперь все встанет на свои места.

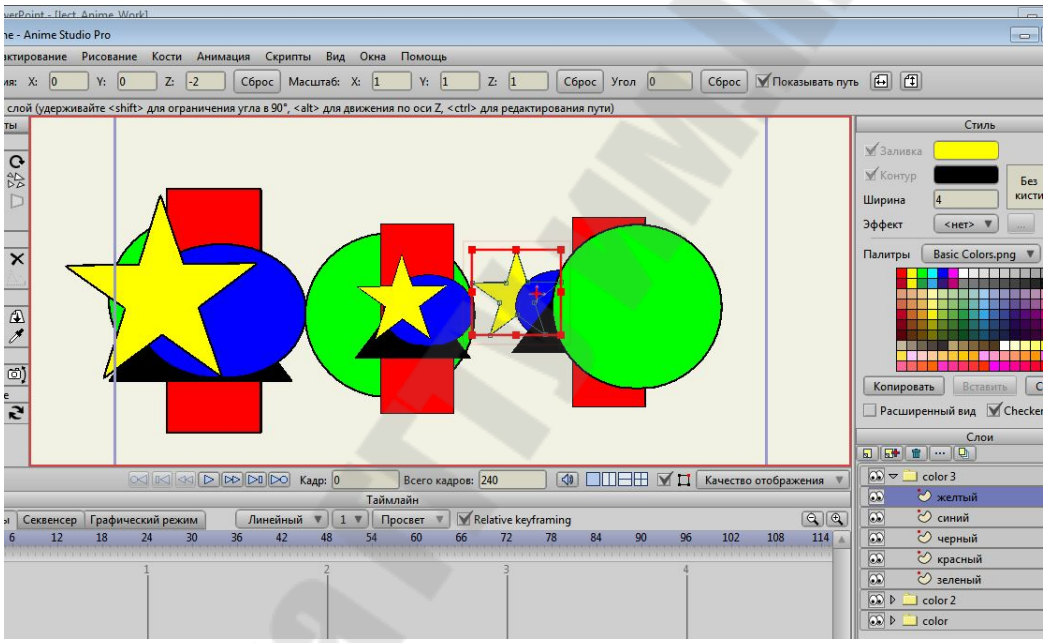

Рисунок 5.2 – Работа z-индексами в Anime Studio

Простая анимация. Полет вращающегося овального мяча

- Перейдем в нулевой кадр
- Рисуем кружок в левом нижнем углу
- Переходим в 96 кадр
- Выделяем значок слоя
- Тащим кружок в правый верхний угол
- Если опция «указывать путь установлена», то видна линия, соединяющая начальную и конечную точки.
- Зажав кнопку Ctrl можно редактировать путь

- Перейдем в 96 кадр
- На палитре инструментов нажмем «Слой»
- Сверху на контекстной панели выставим угол 720
- Вернемся в 0 кадр. Пользуясь панелью таймлайн, запустим анимацию.

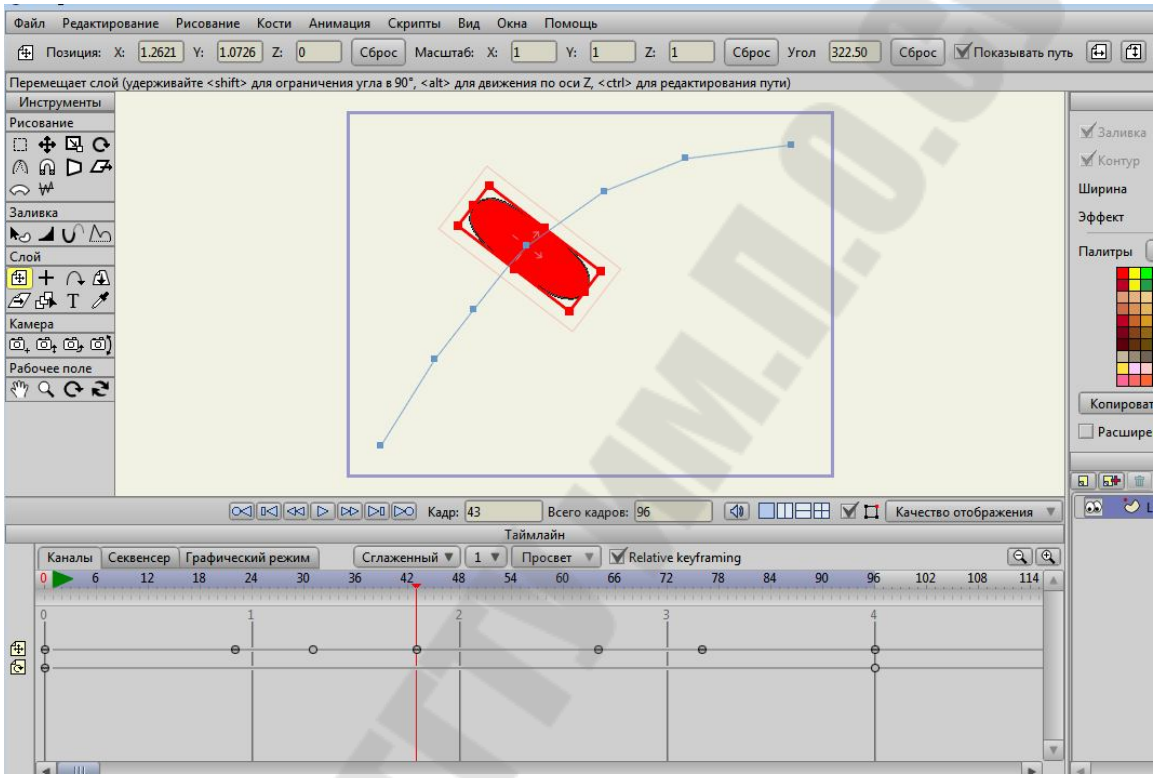

Рисунок 5.3 – Простая анимация в Anime Studio

Простая анимация. Плавное изменение цвета

- Сделаем так, чтобы цвет эллипса в процессе движения поменялся с красного на черный.
- Перейдем в кадр номер 96.
- Щелкнем куда-нибудь на панели рисования.
- На панели заливка кликнем выбрать форму.
- Щелкнем по нашему эллипсу.
- На панели color выберем черный цвет – мы задали черный цвет в последнем кадре.
- Пользуясь панелью timeline, проверим анимацию.

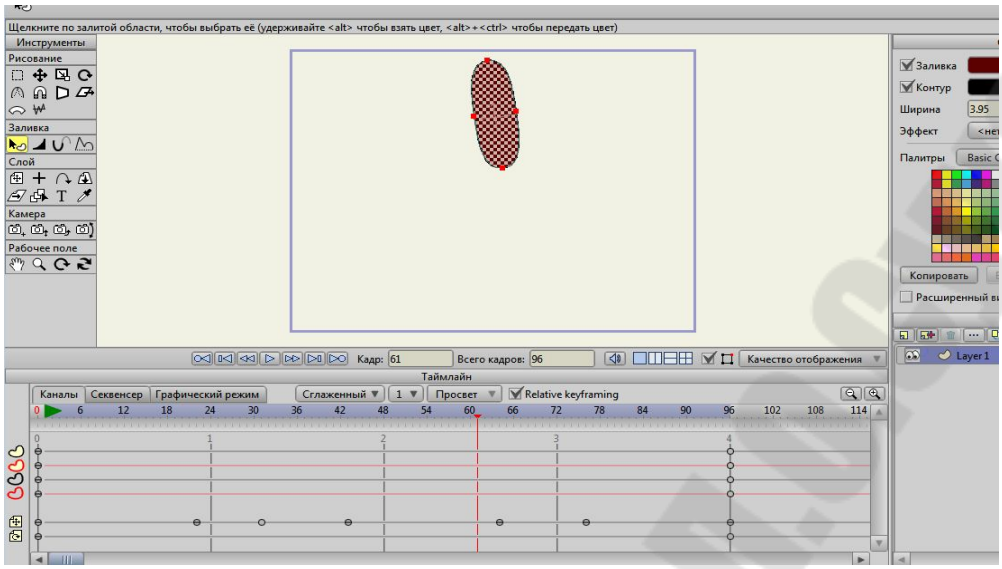

Рисунок 5.4 – Анимация цвета в Anime Studio

В заключение, приведем несколько фрагментов из студенческих работ.

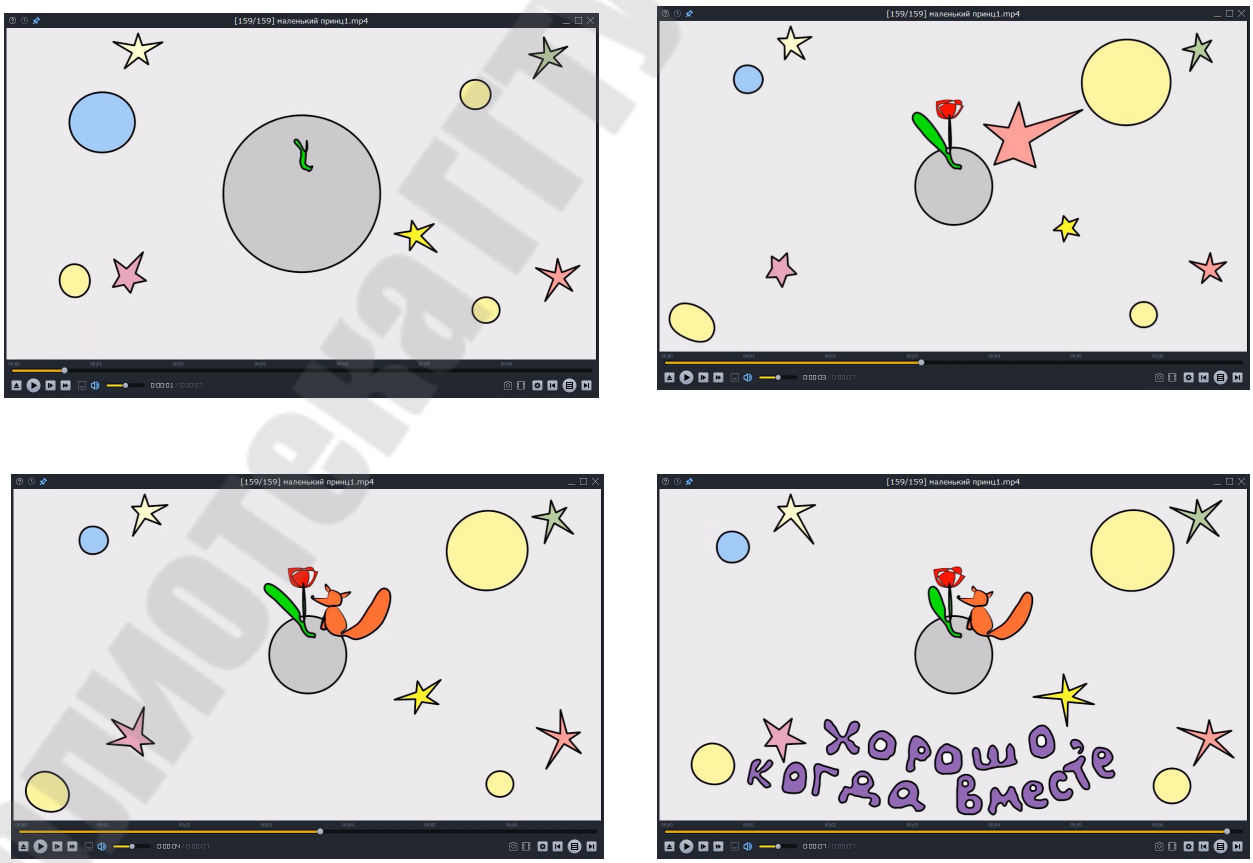

Рисунок 5.5 – Кадры из мультфильма «Маленький Принц» (Дарья Шандрак)

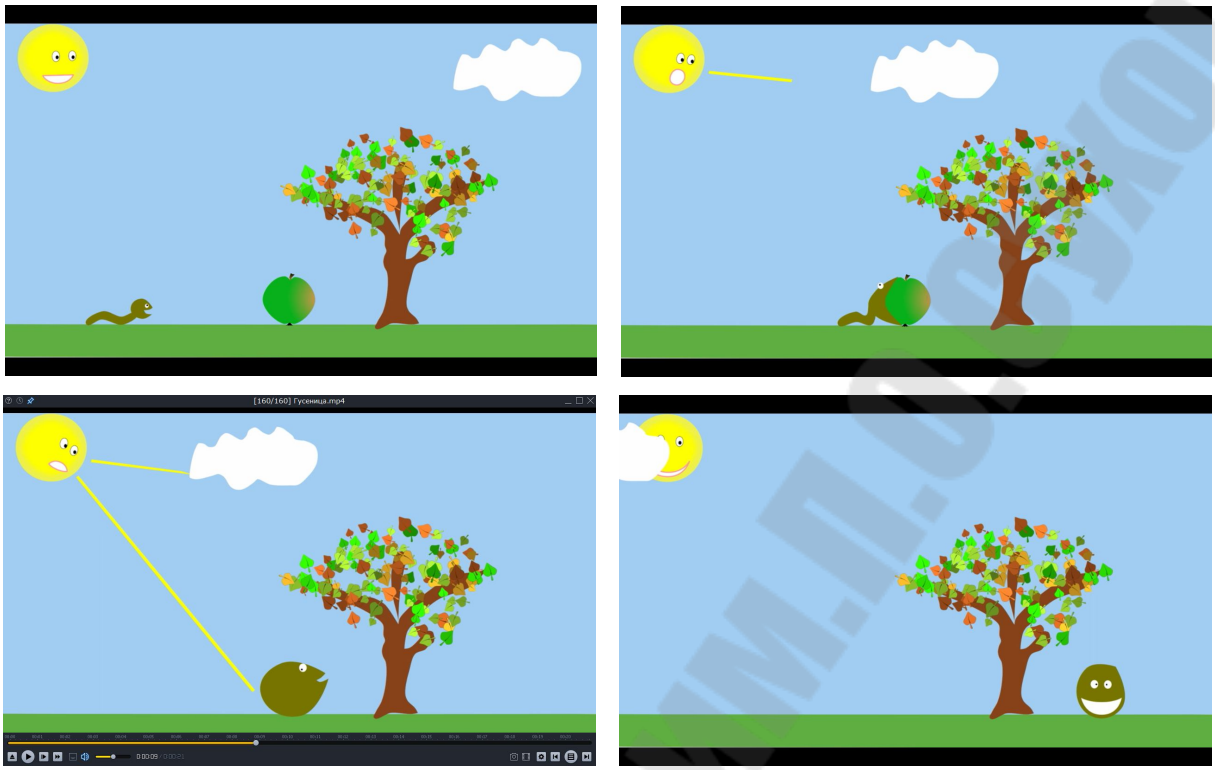

Рисунок 5.6 – Кадры из мультфильма «Гусеница» (Геннадий Миркин)

Задание к лабораторной работе №5

1) продумать сюжет для анимационного ролика длиной от 10 до 20 секунд;

2) написать краткий сценарий будущего мультфильма (1-2 абзаца);

3) создать анимацию в соответствии с разработанным сценарием, используя Anime Studio, или другое приложение, предоставляющее аналогичные возможности;

4) экспортировать ролик в формат mp4 или avi.

Контрольные вопросы

1. Какие виды анимаций вы знаете?

2. Что такое tween-анимация?

3. Что такое cut-out-анимация?

4. Как создать анимацию формы и цвета в Anime Studio?

5. Что собой представляет покадровая анимация? В каких случаях она предпочтительнее tween-анимации?

Бельский Вадим Алексеевич

АНИМАЦИОННАЯ ГРАФИКА

Практикум
по одноименному курсу
для слушателей специальности
1-40 01 74 «Web-дизайн и компьютерная графика»
заочной формы обучения

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 04.01.19.
Рег. № 64Е.
http://www.gstu.by