МЕТОДЫ И АЛГОРИТМЫ ГЕНЕРАЦИИ ЗАЩИТНЫХ ИЗОБРАЖЕНИЙ

А.Н. Щербакова[1)], Н.А. Савчук[2)], О.А. Новосельская[3)], Д.М. Романенко[4)]

1) магистрант факультета информационных технологий Учреждения образования «Белорусский государственный технологический университет», г. Минск, Республика Беларусь, 1717alina1717@gmail.com

2) ассистент Учреждения образования «Белорусский государственный технологический университет», г. Минск, Республика Беларусь, nadezhda.savchuk@gmail.com

3) к.т.н., доцент Учреждения образования «Белорусский государственный технологический университет», г. Минск, Республика Беларусь, nochka@tut.by

4) доцент, к.т.н, заведующий кафедрой Учреждения образования «Белорусский государственный технологический университет», г. Минск, Республика Беларусь, rdm@belstu.by

Аннотация: в статье рассмотрены методы и алгоритмы, используемые для разработки программного продукта по формированию защитного изображения для печати.

Ключевые слова: защитное изображение, алгоритм, программное обеспечение.

METHODS AND ALGORITHMS FOR GENERATION A PROTECTIVE IMAGE

Alina N. Scherbakova[1)], Nadezhda A. Savchuk[2)], Olga A. Novoselskaya[3)], Dmitry M. Romanenko[4)]

1) undergraduate student of Information Technology Faculty, Educational Institution “Belarusian State Technological Unversity”, city of Minsk, Republic of Belarus, 1717alina1717@gmail.com

2) assistant of Educational Institution “Belarusian State Technological Unversity”, city of Minsk, Republic of Belarus, nadezhda.savchuk@gmail.com

3) Ph. D., associate professor, Educational Institution “Belarusian State Technological Unversity”, city of Minsk, Republic of Belarus, nochka@tut.by

4) associate professor, Ph.D., head of the department of Educational Institution “Belarusian State Technological Unversity”, city of Minsk, Republic of Belarus, rdm@belstu.by

Abstract: This article discusses methods and algorithms for designing the software of forming a protective image for its printing.

Key words: protective image, algorithm, software.

С каждым годом необходимость борьбы с фальсификацией становится все более острой, но абсолютной защиты не бывает. Любую вещь, созданную человеком или группой людей, можно повторить. Здесь играют роль экономический и временной факторы. Какими бы сложными и дорогостоящими не были средства защиты продукции от фальсификации, по истечению времени появляется способ их воспроизведения. Поэтому эффективность защиты напрямую зависит от новизны методов.

На сегодняшний день в мире создано уже немало методов защиты от подделок, например, полиграфические, химические, физические, электронные и так далее. Все используемые технические и технологические методы защиты продукции можно разделить на пять больших групп [1]:

1. Защита на стадии дизайна.
2. Технологические способы печати (орловская, офсетная, глубокая печать).
3. Защита за счет использования особенностей бумаги.
4. Защита с помощью специальных красок.
5. Использование финишных и отделочных процедур после печати.

В данной работе предложены методы и алгоритмы генерации векторных защитных элементов, а также разработанный на их основе программный продукт. В основе алгоритмов лежат преобразования базового примитива, такие, как поворот, масштабирование, смещение, а также технология автотипного синтеза цвета. Автотипный синтез цвета – получение оттенков цвета на оттиске путем совмещения растровых или штриховых изображений, отпечатанных красками разных цветов, например, триадными красками: голубой, пурпурной и желтой (CMY) [2]. При рассматривании изображения, полученного автотипным синтезом, на достаточно большом расстоянии происходит пространственное смешение цветов. На рисунке 1 представлено одно и то же изображение в разном масштабе.

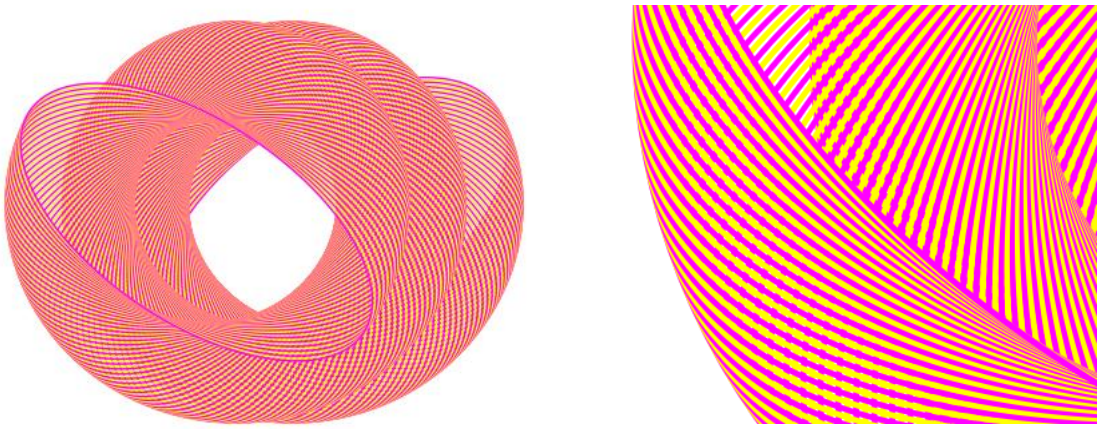

Рисунок 1 – Защитный элемент в разном масштабе воспроизведения

В разрабатываемом программном продукте красный цвет получается путем чередования желтого и пурпурного цветов. Формирование красного цвета можно представить в виде блок-схемы, которая представлена на рисунке 2. Аналогичным образом получается зеленый цвет – чередование желтого и голубого и синий – чередование пурпурного и голубого.

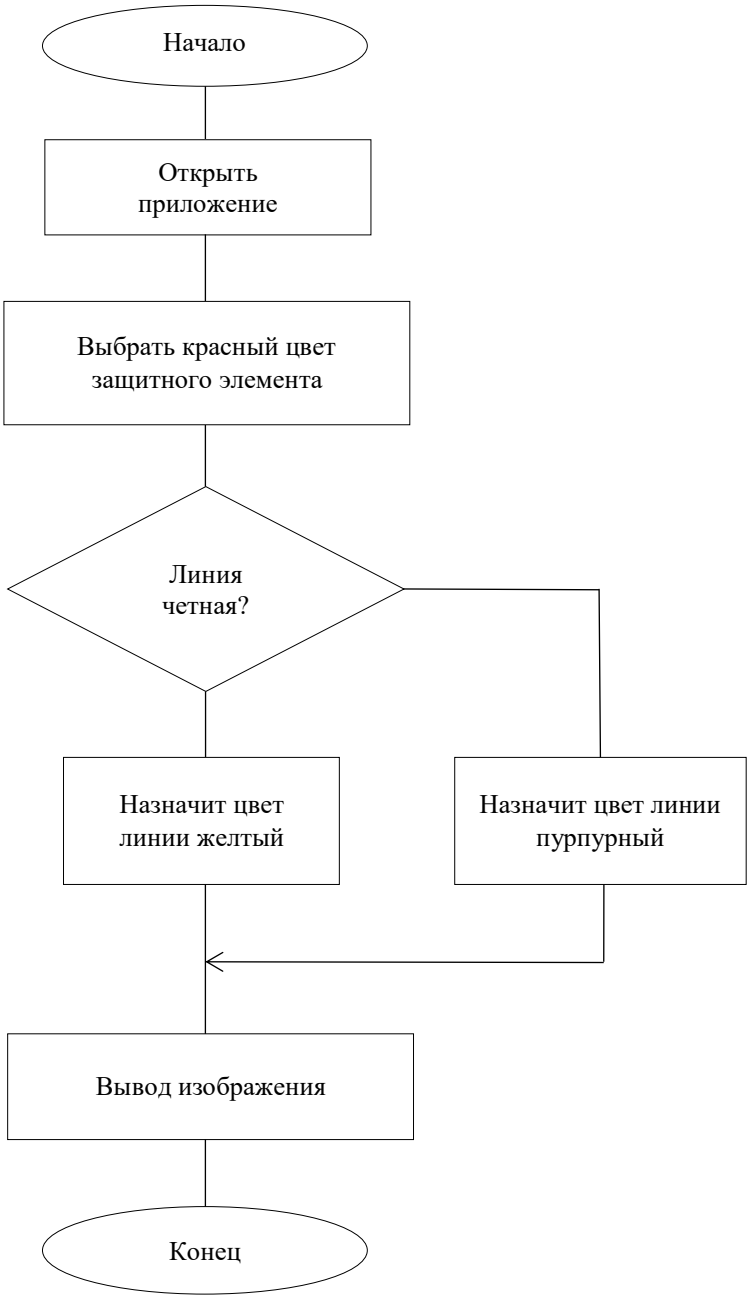

Рисунок 2 – Алгоритм формирования красного цвета

Векторные защитные изображения формируются из простейших геометрических фигур. К простейшим фигурам в данном случае можно отнести эллипс, круг, прямоугольник, квадрат, треугольник, линия, а также произвольная фигура. Элементы формируются путем трансформации вращения с заданным шагом, углом поворота и количеством копий. В результате различных комбинаций операций из одной фигуры можно получить разные формы [3,4]. Программный продукт для их генерации

разработан на языке программирования JavaScript и состоит из следующих классов:

1) класс Image, который хранит параметры изображения, а именно размеры изображения по вертикали и горизонтали, а также цвет фона (либо прозрачный) (листинг 1).

```
class Image {
constructor() {
this.dimensions = {
x: 400,
y: 400
}
this.backgroundColor = "#000000"
this.transparentBackground = false}
}
```

Листинг 1 – Класс Image

2) класс Shape, который хранит информацию о выбранном примитиве и его параметрах (dimensions – размеры по горизонтали и вертикали, transform (translate – смещение по горизонтали и вертикали, scale – масштабирование по горизонтали и вертикали, rotate – поворот), strokeWidth – толщина обводки.

3) класс FinalShape, который содержит параметры результирующего примитива, наследуется от класса примитива Shape.

4) класс Transition, хранящий данные о переходе между примитивами. Содержит shape – тип примитива, customPoints – хранит координаты точек, заданных пользователем, если был выбран произвольный примитив, steps – количество шагов перехода, pivot – опорная точка, относительно которой поворачиваются и масштабируются примитивы.

5) Класс UI, который хранит информацию об элементах интерфейса.

Для изменения примитивов на *i*-шаге реализованы следующие вычисляемые свойства:

1) шаг изменения ширины и высоты примитива – widthDelta() и heightDelta() соответственно (листинг 2).

2) шаг изменения смещения примитива по горизонтали и вертикали – translationXDelta() и translationYDelta() соответственно;

3) шаг масштабирования примитива по горизонтали и вертикали – scaleXDelta() и scaleYDelta() соответственно;

4) шаг поворота примитива rotationDelta();

5) шаг изменения толщины обводки strokeWidthDelta().

```
widthDelta() {
if (!this.transition.changeDimensions)
return 0
```

```
     return (this.shape.to.dimensions.x -
this.shape.from.dimensions.x) / this.transition.steps
     },
     heightDelta() {
     if (!this.transition.changeDimensions)
     return 0
     return (this.shape.to.dimensions.y -
this.shape.from.dimensions.y) / this.transition.steps
     }
```

Листинг 2 – Вычисляемые свойства

Для изменения примитивов на *i*-шаге реализованы следующие методы:

1) вычисление ширины и высоты примитива на *i*-том шаге перехода – widthI(i) и heightI(i);

2) вычисление смещения примитива по горизонтали и вертикали на i-том шаге перехода – translationXI(i) и translationYI(i).

3) вычисление масштабирования примитива по горизонтали и вертикали на i-том шаге перехода – scaleXI(i) и scaleYI(i).

3) вычисление поворота примитива на *i*-том шаге перехода – rotationI(i).

4) вычисление толщины обводки на *i*-том шаге перехода – strokeWidthI(i).

Генерация изображения начинается с ввода размера холста, можно выбрать фоновый цвет либо сделать фон прозрачным. Основой алгоритма является выбор базового примитива: круг, эллипс, прямоугольник, квадрат, треугольник, линия либо произвольный примитив. При выборе произвольного примитива необходимо ввести координаты точек для его построения. Далее алгоритм предполагает ввод результирующего цвета. При выборе красного цвета четные примитивы будут иметь желтую обводку, а нечетные – пурпурную, при выборе зеленого цвета четные примитивы будут иметь голубую обводку, а нечетные – желтую, при выборе синего цвета четные примитивы будут иметь голубую обводку, а нечетные – пурпурную. Далее необходимо ввести параметры начального и конечного размера для выбранного базового примитива, а также масштаб, поворот и толщину обводки. По умолчанию в качестве опорной точки выступает центр, но при желании опорную точку можно изменить, введя параметры x и y.

Следующим этапом задается количество шагов. Если данного количество достаточно для формирования защитного изображения, то происходит расчет изменения выбранного примитива на *i*-том шаге

и завершение генерации элемента. Если же количества шагов недостаточно, то необходимо ввести новое значение, при этом происходит перерасчет изменения примитива и генерация нового изображения.

Алгоритм генерации векторного защитного элемента представлен на рисунке 3.

Изображение можно сохранить в формате SVG и при необходимости модифицировать его в любом графическом редакторе. Для сохранения изображения в формате EPS был добавлен модуль на языке описания страниц PostScript.

Далее рассмотрим пример генерации защитного изображения с использованием базового примитива прямоугольник в разработанном приложении.

Для создания изображения были выбраны следующие параметры:

1) тип примитива: прямоугольник;

2) количество шагов: 250;

3) желаемый результирующий цвет: красный;

4) размеры (x, y): начальное значение (250;300), конечное значение (50;100);

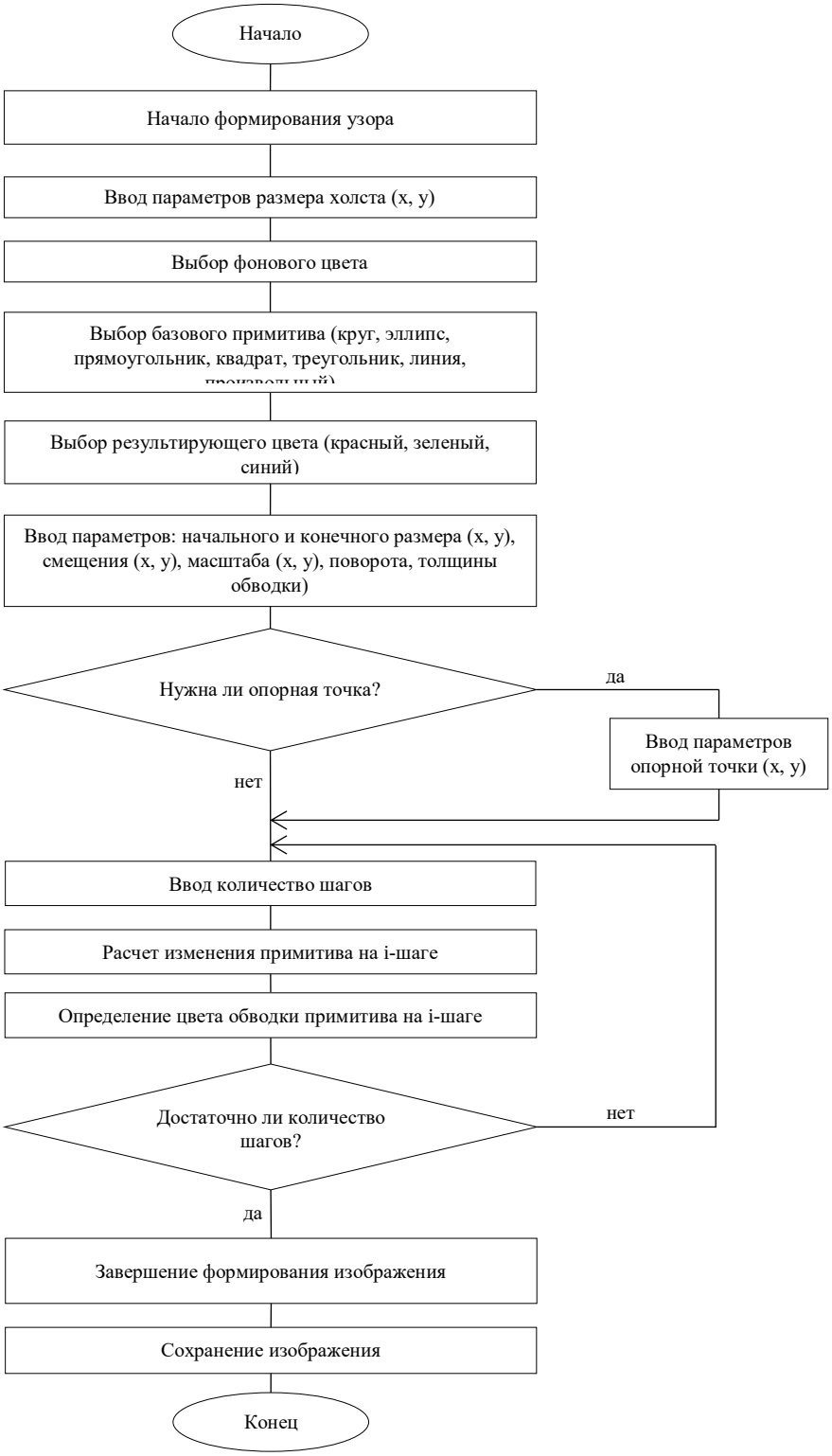

Рисунок 3 – Алгоритм генерации защитного изображения

5) смещение (x, y): начальное значение (0; 0), конечное значение (0; 0);

6) масштаб (x, y): начальное значение (1; 1), конечное значение (1; 1);
7) поворот: начальное значение (0), конечное значение (180);
8) толщина обводки: начальное значение (3), конечное значение (0);
9) цвет фона: прозрачный.

Вариант защитного изображения представлен на рисунке 4.

Рисунок 4 – Вариант защитного изображения

Для изображений был выбран формат SVG, который позволяет с легкостью масштабировать полученные объекты без потери качества, в отличие от растровых изображений, которые при увеличении масштаба получаются зернистыми и размытыми. Модифицировать SVG-файл можно как при помощи графических редактором, так и с помощью CSS непосредственно на сайте. Можно с легкостью изменить цвет, форму изображения. Также преимуществом является возможность добавлять анимацию с помощью JavaScript либо CSS3. Файлы данного формата занимают меньше места, чем растровые картинки, что очень важно для скорости загрузки веб-страниц.

Список использованных источников:

1. Дубина, Н. Полиграфические методы защиты [Электронный ресурс] / Н. Дубина // КомпьюАрт. – № 1. – 2002. – Режим доступа: https://compuart.ru/article/ 8348. – Дата доступа: 15.02.2016.

2. Стефанов, С. Полиграфия от а до Я: энциклопедия. – М.: URSS, 2009.

3. Горовенко А.Д., Хамдан Х.М., Горовенко Л.А. Использование рекурсивных изображений в архитектуре и дизайне // Прикладные вопросы точных наук Материалы III Международной научно-практической конференции студентов, аспирантов, преподавателей. - Армавир: РИО АГПУ, 2019. - С. 37-41.

4. Груднов И.А., Горовенко Л.А. исследование методов программирования фрактальных изображений // Прикладные вопросы точных наук Материалы III Международной научно-практической конференции студентов, аспирантов, преподавателей. - Армавир: РИО АГПУ, 2019. - С. 248-252.