УДК 004.9

Автоматизированное построение непрерывных текстур

А. В. Сенина

Одесский национальный университет им. И. И. Мечникова, Украина

В данной работе исследуется задача создания текстур (или мозаик) в векторном и растровом случаях. В первой части статьи предлагается реализация алгоритма интерактивного построения мозаик, который базируется на работах М. К. Эшера. Во второй части изучается создание текстур на основе произвольных растровый изображений. Для этого разрабатывается аналитический подход и предлагается его дискретная реализация с использованием динамического программирования и методов оптимизации. Существует широкое практическое применение таких мозаик и текстур – в web-дизайне, в качестве обоев, текстур поверхностей для 3D-объектов, в компьютерных играх и др. Данные задачи представляют и теоретический интерес, они активно исследуются.

Ключевые слова: *текстура, мозаика, алгоритм интерактивного построения, динамическое программирование, оптимизация, web-дизайн.*

У даній роботі досліджується задача створення текстур (або мозаїк) у векторному та растровому випадках. У першій частині статті пропонується реалізація алгоритму інтерактивної побудови мозаїк, що базується на роботах М. К. Ешера. У другій частині вивчається створення текстур на основі довільних растрових зображень. Для цього розроблюється аналітичний підхід та пропонується його дискретна реалізація із використанням динамічного програмування та методів оптимізації. Існує широке практичне застосування таких мозаїк та текстур – у web-дизайні, у якості шпалер, текстур поверхонь для 3D-об'єктів, у комп'ютерних іграх та ін. Також, дані задачі становлять теоретичний інтерес, і останніми роками активно досліджуються.

Ключові слова: *текстура, мозаїка, алгоритм інтерактивної побудови, динамічне програмування, оптимізація, web-дизайн.*

In this work some problems of vector and raster texture (mosaic) generation are investigated. In the first part of the paper, we propose a realization of the mosaic construction interactive algorithm, which is based on M. C. Escher's works. In the second part, we explore the creation of textures based on arbitrary raster images. Thereto, an analytical approach is developed and its discrete realization, with the dynamic programming and optimization methods usage, is proposed. There is wide practical use of such mosaics ant textures – in web-design, as wallpapers, texture surfaces for 3D-objects, in computer games etc. Also, these problems are of theoretical interest and have been widely investigated last years.

Keywords: *texture, mosaic, construction interactive algorithm, dynamic programming, optimization, web-design*

1. Постановка задачи и её актуальность

Рассмотрим задачу о замощении плоскости замкнутыми фигурами без перекрытий и щелей. Будем называть такое замощение мозаикой (или паркетом) и рассматривать случай заполнения плоскости многоугольниками. Мозаика, состоящая из некоторых повторяющихся фрагментов, называется регулярной. Как известно, из всех правильных многоугольников, регулярно замостить плоскость можно только треугольниками, квадратами и шестиугольниками [1]. Представляет интерес возможность замостить плоскость неправильными многоугольниками. Практическая применимость подобных мозаик широка: многочисленным сайтам в сети Internet требуется оригинальный фон,

дизайнерам моды и интерьеров – узоры, мозаики, витражи и т.д. Идея рассмотрения этой задачи почерпнута из работ М.К. Эшера[1].

Перейдем к следующей задаче. Рассмотрим произвольное растровое изображение. Теперь нам будет необходимо на его основе сгенерировать повторяющуюся текстуру, которая была бы визуально непрерывной, т.е. не имела бы видимых стыковочных границ. В некотором смысле от дискретного случая, описанного выше, мы переходим к непрерывному случаю. Такие изображения в дальнейшем также могут использоваться в качестве фонов web-страниц, обоев, поверхностей для 3D-объектов, в компьютерных играх, а также для многих других дизайнерских целей. Также данная задача представляет теоретический интерес – как частный случай общих задач согласования сигналов, т.к. тут рассматривается вопрос о согласовании краев изображения.

Как видим, выделяются два направления исследований, из которых первое относится к векторной графике, а второе – к растровой, но в целом, они описывают общую задачу. Соответственно для каждого из случаев будут применяться специфические инструменты и методы.

2. Обзор существующих разработок и нерешенные проблемы

2. 1. Методы и алгоритмы построения мозаик

Наиболее полное представление о разнообразии мозаик можно составить, если ознакомиться с рисунками нидерландского художника М.К. Эшера [11, 12]. В творчестве художника можно увидеть визуализацию многих положений теорий цветовой симметрии и симметрии кристаллов. Симметрией в мозаиках принято считать неизменность (инвариантность) её структуры относительно таких преобразований, как поворот, зеркальное отражение и трансляция [2].

Из многочисленных эскизов видно, что основой мозаик служили элементарные фигуры искаженные так, чтобы структура изображения преставала быть узнаваемой. Эшер применял к ним симметричные трансформации, при которых сохранялось главное свойство фигур – способность заполнить плоскость. Так мозаики Эшера приобретали вид птиц, рыб, животных и прочих существ [3]. После внимательного ознакомления с рисунками несложно получить алгоритм их построения. Выбирается основа для создания мозаики – базовая фигура. На этой фигуре изображается требуемый мотив, с использованием соответствующих преобразований (поворот, трансляция, отражение или их комбинация). Фигуры, полученные таким образом, обязательно будут «стыковаться» друг с другом, что и даст требуемое замощение плоскости. Далее, его следует раскрасить.

Коротко коснемся классификации мозаик, составленной Эшером в 1941-1942 годах [4]. Первый фактор классификации – количество используемых цветов. Согласное ему мозаики делятся на двухцветные и трёх- (и более) цветные. Двухцветные мозаики делятся на 5 категорий: A – параллелограммы, B – ромбы, C – прямоугольники, D – квадраты и E – равнобедренные прямоугольные треугольники. Для каждой из категорий может быть выбран идентификатор I, II, …, X, отвечающий типу симметрии данного разбиения. Например, идентификатор I означает: возможность трансляции в обоих поперечных

[1] *Moris Cornelis Escher, 17.06.1898 – 27.03.1972*

(изменяющих цвет) и обоих диагональных (сохраняющих цвет) направлениях. Каждый рисунок, таким образом, отмечался по принципу: IV-B, X-D и т.д.

Исследования Эшера имели систематический характер, созданная им классификация является практически полной. Точные исследования по данной теме были проведены только в последнее двадцатилетие. Очевидно, что Эшер предугадал и прочувствовал многие идеи, касающиеся теории цветовой симметрии. Практически не существует полнофункциональных реализаций, позволяющих наглядно продемонстрировать идеи, заложенные Эшером в его работы. В то же время такие приложения внесли бы существенный вклад в круг прикладных дизайнерских инструментов.

2.2. Существующие программные средства и алгоритмы создания непрерывных текстур

Проанализируем существующие программные средства построения непрерывных текстур на примере одного из таких приложений. TextureWorkshop – это пакет инструментов, который позволяет быстро превратить «любое» изображение в непрерывную повторяющуюся плитку [11]. Тем не менее, на вход программе желательно подавать изображения с некоторыми однородными объектами (листья, камни, бобы и т.д.), тогда результат будет наилучшим. Как видно из результатов работы программы, необходимый эффект достигается путем простого перекрытия изображений – наложения края одного изображения на другое с использованием прозрачности и последующего размытия.

Была сделана попытка анализа существующих алгоритмов создания текстур на основе изображений. Такой анализ затрудняется тем, что лишь немногие алгоритмы являются открытыми. Итак, чаще всего применяются следующие методы: разнообразные волновые сглаживание границ изображения и частичное перекрытие изображений (см. выше). Эти простые алгоритмы способны давать приемлемый результат, но все же часто оказываются слишком примитивными.

Существует близкая проблема, для решения которой в последние годы придумано множество эффективных алгоритмов. Имеется в виду следующая задача. На вход алгоритма подается малый фрагмент некоторой текстуры, на основе которого следует сгенерировать новую стохастичную текстуру произвольного размера. Большинство решений базируется на генерировании по исходной текстуре новых небольших фрагментов, и в дальнейшей их композиции в одну большую текстуру. В данных решениях широко применяются плитки Ванга [5] и их модификация – ω-плитки [6]. Эти плитки часто упоминаются в контексте задачи замощения плоскости. Они представляют собой квадраты с окрашенными сторонами, соседние плитки должны иметь края одного цвета. Доказано, что такими плитками можно осуществить не только периодические, но и апериодические заполнения плоскости. Существуют также другие решения задачи построения апериодических текстур, такие как алгоритм Эфроса и Фримена [7]. Нужно отметить, что все эти алгоритмы обладают достаточными теоретическими базами и обоснованиями, но относятся к близкой, однако все же отличающейся задаче.

Итак, представляет интерес разработка теоретически обоснованного алгоритма решения поставленной задачи. На сегодняшний момент автору неизвестны подобные решения.

3. Постановка цели и задач исследования

Итак, целью работы является создание алгоритмов построения мозаик (текстур) в векторном и растровом случаях, а также их реализация.

В случае векторных изображений следует разработать программное средство для построения регулярных мозаик на основе элементарных фигур и обеспечить хранение модифицированных базовых фигур для дальнейшего их использования. Необходимо решить следующие задачи:

- проанализировать работы М.К. Эшера и вывести общий принцип создания регулярных мозаик из неправильных многоугольников;
- разработать программное обеспечение, наглядно демонстрирующее описанный принцип.

В случае растровых изображений конечной целью является создание алгоритма, осуществляющего трансформацию произвольного изображения в такое, которое бы без видимых стыков укладывалось «плиткой». Для достижения цели необходимо проделать следующее:

- изучить существующие способы решения данной задачи, проанализировать их достоинства и недостатки;
- разработать собственный алгоритм решения задачи, предложить его математическое обоснование и формализацию;
- создать программное обеспечение, реализующее данный способ решения задачи.

4. Описание разработки программной системы построения мозаик

4.1. Алгоритм и структуры данных

Введём в рассмотрение два важных понятия. **Симплекс** – элементарная базовая фигура, которая лежит в основе мозаичной структуры. **Комплекс** – результат замощения плоскости симплексом. Основная идея решения – для построения комплекса необходимо специальным образом сформировать симплекс [8]. Он будет храниться в памяти компьютера в виде массива точек `points` – вершин многоугольника.

Пользователь имеет возможность деформировать стороны многоугольника (при помощи мыши), «вытягивая» новые вершины. Также в памяти хранится список соответствующих сторон `pointList`, в виде: `(i,j)~(k,l)`, где `i`, `j`, `k`, `l` – номера вершин многоугольника. При изменении одной из сторон многоугольника, наблюдаем одновременное изменение соответствующей ей стороны (см. рис. 4.1). Таким образом, получаем две новые вершины. Далее, дополняем массив `points` двумя новыми точками. Из списка `pointList` необходимо удалить указанную пару сторон и добавить вместо неё две новые пары. Например, при условии, что общее количество вершин равно `n`, получим новые пары сторон: `(i,n)~(k,n+1)` и `(n,j)~(n+1,l)`. Для данных симплексов (параллелограмм, правильный шестиугольник) соответствующие стороны параллельны, что облегчает расчёт координат второй точки (тут и далее – точка, которая образуется автоматически, без участия пользователя). Воспользуемся критерием параллелограмма: пусть, точка, в которой пользователь отпустил мышь – `Up(Up.X; Up.Y)`, тогда координаты новой точки `New`:

```
New.X = points[k].X + Up.X – points[i].X      (4.1)
New.Y = points[k].Y + Up.Y – points[i].Y      (4.2)
```

Ясно, что для реализации, например, мозаики с поворотом, необходимо будет только изменить способ выбора соответствующей стороны и способ расчёта координат второй точки.

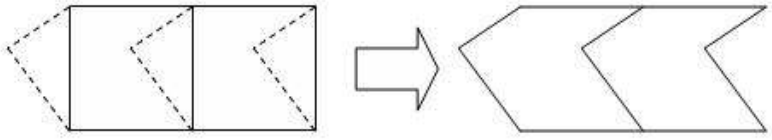

Рис. 4.1. При изменении одной стороны – изменяется соответствующая ей, таким образом, снова получаем стыкующиеся фигуры

4.2. Реализация программного обеспечения и полученные результаты

В разработанном приложении используется язык программирования C# и среда разработки .Net. Была создана иерархия наследования для классов, представляющих различные типы симплексов. Также была реализована XML-сериализация объектов базового класса. Для осуществления сериализации в .Net существует пространство имён System.Xml.Serialization.

Ниже представлены некоторые примеры работы приложения.

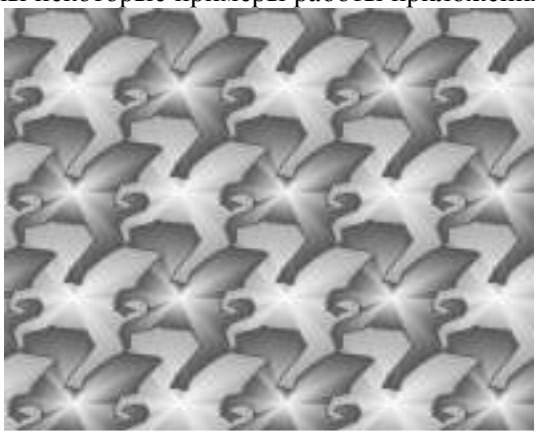

Рис. 4.2. Мозаика на основе квадратов, 2 цвета

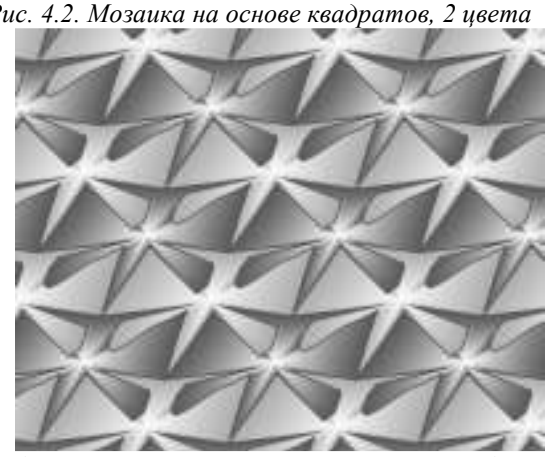

Рис. 4.3. Мозаика на основе параллелограммов, 2 цвета

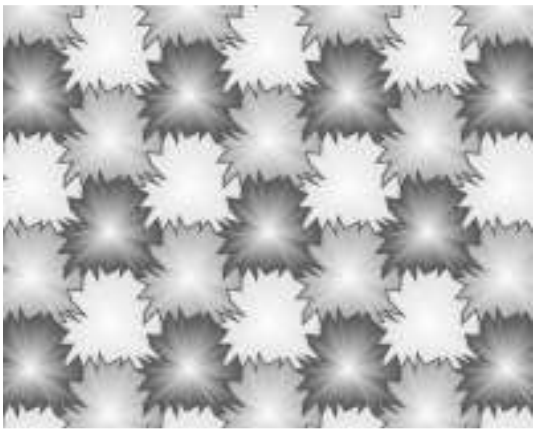

Рис. 4.4. Мозаика на основе шестиугольников, 3 цвета

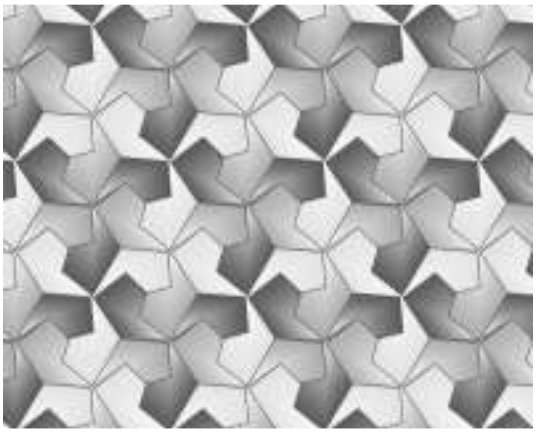

Рис. 4.5. Мозаика на основе шестиугольников, 3 цвета

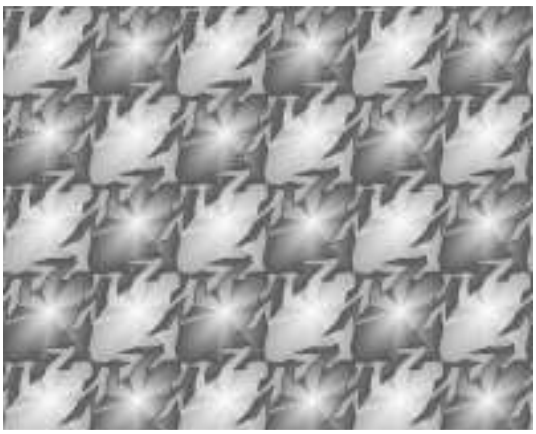

Рис. 4.6. Мозаика на основе квадратов, 2 цвета

5. Описание разработки программной системы построения непрерывных текстур

5.1. Теоретическое обоснование алгоритма

Как возможное решение поставленной задачи, предлагается генерировать изображения с гладкими функциями переходов на границах [9]. А именно, если некоторая функция f задана на отрезке $[a;b]$, то для обеспечения гладкости перехода (продолжения) должны выполняться два условия:

$$f(a)=f(b), f'(a)=f'(b). \tag{5.1}$$

Несложно понять, что изображения, удовлетворяющие таким условиям, были бы искомыми. Поставим целью построить алгоритм, который на вход принимает произвольное изображение, а на выход выдает новое, модифицированное с целью получения гладких функций переходов на границах.

Рассмотрим изображение размером 1×1. Пусть $\overline{f(x)}=\overline{(f_1(x),f_2(x),f_3(x))}$ и $h(x)=\overline{(h_1(x),h_2(x),h_3(x))}$ – функции цвета на отрезках OC и AB, соответственно (это вектор-функция из трёх компонент – Red, Green, Blue).

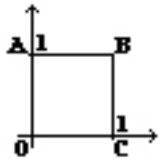

Рис. 5.1 Изображение OABC размером 1×1

Запишем условия гладкой «стыковки» по вертикали, аналогичные (5.1):

$$f_i(x)=h_i(x), f_i'(x)=h_i'(x), i=\overline{1,3} \tag{5.2}$$

Так же можно ввести условия по горизонтали.

На данном этапе поставим задачу так: необходимо наложить такое преобразование на вектор-функцию $\overline{f(x)}$ (вернее на отрезок *OC*), чтоб максимально приблизить значения $\overline{f(x)}$ к $\overline{h(x)}$. То есть, если F – это некоторый функционал, то один из путей решения задачи – минимизация функционала:

$$\int_0^1 (f_i(F(x))-h_i(x))^2\,dx \tag{5.3}$$

или, если это возможно, его оценка сверху:

$$\int_0^1 (f_i(F(x))-h_i(x))^2\,dx<\varepsilon \tag{5.4}$$

где ε достаточно мало, $i=\overline{1,3}$. Как видим, суть трансформационного преобразования функции $\overline{f(x)}$ сводится к изменению её области определения. Вспомним, что на самом деле изображение состоит из дискретного набора пикселей. Тогда становится ясно, что преобразование является деформацией граничного слоя пикселей (растяжениям, сжатиям и т.д.) с целью приближения его к другому слою.

Наложим некоторые условия на функционал F. Во-первых, пусть он будет монотонным (не ограничивая общности – монотонно возрастающим), для того чтоб преобразование было взаимно однозначным. Во-вторых, будем искать, если это возможно, функционал, который наименее уклоняется от тождественного $P(x) \equiv x$. Если ввести норму как:

$$\int_0^1 (F(x) - x)^2 \, dx \tag{5.5}$$

то искомый функционал будет минимальным по норме. Это условие вытекает из необходимости как можно меньше деформировать изображение во избежание потери качества и появления дефектов. И, наконец, граничные условия:

$$F(0) = 0, F(1) = 1 \tag{5.6}$$

гарантируют отображение отрезка $[0;1]$ «в себя».

Сначала мы пытались формализовать задачу в наиболее общеё форме, теперь сузим её. Для этого ограничим круг исследуемых функционалов. Тут предпринимается попытка подбора подходящего функционала из некоторого специального класса. А именно, рассмотрим функционалы, которые являются многочленами и удовлетворяют условию:

$$F'(x) = \prod_{i=1}^{n} (x - x_i)^{2k_i} \tag{5.7}$$

Производная $F'(x) \geq 0$ и, следовательно, монотонность F обеспечена.

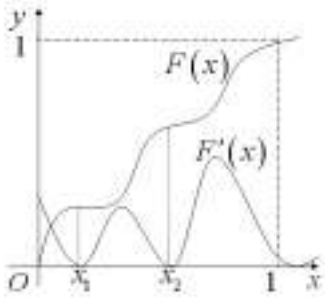

Рис 5.2. Схематическое представление функционала и его производной

Такой выбор функционала объясним наглядностью и гибкостью преобразования отрезка $[0;1]$ (см. рис. 5.2). А собственно минимизация расхождения между правой и левой границей будет достигаться путем подбора корней производной x_i и их кратностей $2k_i$. Итак, требуется найти общий вид описанного функционала.

Решим похожую задачу. Найдем функционал $\overline{F}$, для которого:

$$\overline{F}'(x) = \prod_{i=1}^{N} (x - x_i) \tag{5.8}$$

По обобщенной теореме Виета:

$$\overline{F}'(x) = \prod_{i=1}^{N} (x - x_i) = x^N - (x_1 + x_2 + \ldots + x_N) x^{N-1} +$$

$$+\left(x_1x_2+x_1x_3+\ldots x_{N-1}x_N\right)x^{N-2}-\ldots+\left(-1\right)^N x_1x_2\ldots x_N. \qquad (5.9)$$

$$\overline{F}(x)=\frac{1}{N+1}x^{N+1}-\frac{1}{N}\left(x_1+x_2+\ldots x_N\right)x^N+$$

$$+\frac{1}{N-1}\left(x_1x_2+x_1x_3+\ldots+x_{N-1}x_N\right)x^{N-1}-\ldots+\left(-1\right)^N x_1x_2\ldots x_Nx. \qquad (5.10)$$

Отметим, что граничное условие $\overline{F}(0)=0$ выполнено. Для выполнения условия $\overline{F}(1)=1$ просто представим $\overline{F}(x)$ в виде $\frac{\overline{F}(x)}{\overline{F}(1)}$.

Таким образом, получена общая формула для $\overline{F}$. Чтобы получить общий вид функционала F, удовлетворяющего формуле (5.7), достаточно рассмотреть $\overline{F}$, в котором точки x_i повторяются необходимое число раз, а именно $2k_i$.

Тем не менее, в общем случае задача является громоздкой и трудоемкой. Подсчет значения функционала $F(x)$ при заданных значениях x_i и k_i, был реализован программно и был получен неудовлетворительный по времени результат. Стало ясно, что реализация описанных выше рассуждений напрямую, является излишне ресурсоемкой. Возникает необходимость в поиске алгоритма, реализующего те же идеи, но за приемлемое время.

5.2. Алгоритм и структуры данных

Ещё раз заметим, что описанные трансформации, в общем, сводятся к изменению «ширины» пикселей граничного слоя. Проблему составляет именно оптимальный выбор такой ширины для каждого пикселя. Покажем, что данная задача может быть сформулирована в виде модели динамического программирования [10].

Будем считать, что поставленная задача удовлетворяет принципу оптимальности Беллмана, а именно, что оптимальное распределение пикселей по ширине для всего изображения в целом, будет оптимальным для любой его части.

Итак, в начале необходимо выделить основные элементы модели. Пусть имеется изображение шириной m пикселей. Следует разбить отрезок $[0;1]$ на m частей $0\le y_i\le 1,\ \ i=\overline{0,m-1}$, так чтоб величина некоторого функционала была минимальной. Этапом i, очевидно, будет выступать номер обрабатываемого пикселя. Состоянием x_i на этапе i будет длина уже заполненной за предыдущие этапы части отрезка $[0;1]$. На каждом этапе в качестве управления выбирается соответствующая ширина y_i. Ограничения на управления y_i имеют вид:

$$\sum_{i=0}^{m-1} y_i = 1 \qquad (5.11)$$

Пусть начальная ширина всех пикселей – Δh. Введем коэффициент разбиения пикселя $C\in \mathrm{N}$, и соответственно определим шаг изменения

управления, как $\Delta y = \frac{\Delta h}{C}$. Чем больше C, тем более тщательно выбираются оптимальные значения. Но при этом значительно увеличивает время работы алгоритма, поэтому на этот параметр поставлено ограничение $C = \overline{1,10}$. Приемлемое время работы и качество достигаются в среднем при $C = \overline{3,5}$.

Таким образом, на каждом этапе управление y_i выбирается как одно из возможных:

$$0, \Delta y, 2\Delta y, \ldots, 1 \tag{5.12}$$

Нам известно каким должно быть состояние на последнем этапе, а именно: $x_{m-1} = 1$. Связь между соседними состояниями выглядит так:

$$x_i = x_{i+1} - y_{i+1},\ i = \overline{0, m-1} \tag{5.13}$$

Теперь обсудим, какой именно функционал будет минимизироваться. Для совмещения требований по минимизации ошибки преобразования (5.3) и нормы преобразования (5.5), введем следующий функционал:

$$\alpha \int_0^1 \left(F(x) - x\right)^2 dx + \beta \int_0^1 \left(f_i(F(x)) - h_i(x)\right)^2 dx, \tag{5.14}$$

где α, β – коэффициенты (между 0 и 1), определяемые пользователем в каждой конкретной ситуации по степени важности данных слагаемых.

Наконец, перейдем к дискретному представлению. Тут появятся небольшие изменения по сравнению с непрерывным случаем, что обусловлено особенностями метода динамического программирования.

Имеем окончательный вид минимизируемого функционала:

$$\alpha \sum_{i=1}^{m-1} \left(\Delta h - y_i\right)^2 + \beta \sum_{i=1}^{m-1} \sum_{j=x_i - y_i}^{x_i - \Delta y} \left(\left(f\right)_i - \left(h\right)_{ij} \right)^2, \tag{5.15}$$

где $\left(f\right)_i$ – значение цвета в i-том пикселе на нижней границе, шаг изменения j равен Δy, $\left(h\right)_{ij}$ – значение цвета в j-том участке «нового» i-го пикселя на верхней границе изображения. Нужно отметить, что значение $\left(f\right)_i$ определяется просто, как оригинальный цвет в данном пикселе. В отличие от него, значения $\left(h\right)_{ij}$ находятся по оригинальному цвету лишь в пределах ширины y_i с учетом состояния x_i. Таким образом, сумма квадратов по каждому пикселю разбивается ещё и в сумму квадратов по каждой части пикселя Δy.

Осталось записать уравнения Беллмана для описанной модели.

$$w_0(x_0) = \min_{0 \le y_0 \le 1} \left\{ \alpha \left(\Delta h - y_i\right)^2 + \beta \left(\left(f\right)_0 - \left(h'\right)_0 \right)^2 \right\} \tag{5.16}$$

$$w_i(x_i) = \min_{0 \le y_i \le 1} \left\{ \alpha \left(\Delta h - y_i\right)^2 + \beta \left(\left(f\right)_i - \left(h'\right)_i \right)^2 + w_i\left(x_i - y_i\right) \right\} \tag{5.17}$$

В формулу (5.16), как видим, уже подставлена рекуррентная формула для состояний (5.13).

В данной задаче управления и состояния тесно связаны, что приводит к некоторым изменениям по сравнению с общей схемой решения. Собственно изменения выражены в том, что на этапе $i = 0$ управление никак не может

отличаться от состояния, т.е. всегда $x_0 = y_0$. В остальном стандартная схема прямой прогонкой остается в силе.

Традиционно при реализации данной процедуры строятся матрицы Fk размерностью $K \times K$, где $K = mC + 1$ – это количество всех возможных управлений из (5.12). Эти же значения фигурируют в качестве допустимых состояний. Реализуется цикл, в котором для каждого обрабатываемого пикселя подсчитывается K^2 значений искомого функционала. Соответственно эти матрицы заполняются, а также заполняются матрицы оптимальных значений функционала Fk_{opt} и управлений Yk_{opt} для каждой итерации цикла. Далее, для последней матрицы извлекается значение оптимального управления, соответствующее состоянию $x_{m-1} = 1$, и во время обратного хода вычисляются все оптимальные управления. Таким образом, вся процедура метода динамического программирования требует введения трех массивов, одного трехмерного $Fk[m][K][K]$ и двух двумерных: $Fk_{opt}[m][K]$ и $Yk_{opt}[m][K]$.

Пусть решается задача вертикальной стыковки, т.е. оптимизируется нижняя граница изображения. Параллельно с вычислением новой границы решается ещё одна, более общая задача. На самом деле, изменив только один ряд пикселей, мы лишь отодвинем проблему гладкой стыковки на один ряд и не более того. Становится ясно, что необходимо изменить всё изображение. Применим к изображению линейное преобразование, как бы выводящее соответствующие пиксели верхней и нижней границы на одну прямую. Ясно, что тогда верхняя часть будет модифицироваться меньше и чем ближе к нижней границе, тем большими будут модификации.

Пусть $w_j(i)$ – искомая ширина j-го пикселя в i-ой строке. В общем виде уравнение имеет вид:

$$w_j(i) = ai + b. \tag{5.18}$$

Из условия имеем, что $\Delta h = w_j(0) = a \cdot 0 + b = b$, откуда $b = \Delta h$. С другой стороны, $dx[j] = w_j(im\_heigth - 1) = a(im\_heigth - 1) + b$, откуда:

$$a = \frac{dx[j] - \Delta h}{im\_heigth - 1}. \tag{5.19}$$

Получаем уравнение прямой:

$$w_j(i) = \frac{dx[j] - \Delta h}{im\_heigth - 1} i + \Delta h. \tag{5.20}$$

Данное уравнение фигурирует в формуле для пересчета ширины пикселей в зависимости от номера «строки» изображения. Таким образом, для всех строк алгоритм пересчета будет совершенно одинаковым, с учетом номера строки.

5.3. Реализация программного обеспечения и полученные результаты

В разработанном приложении используется язык программирования Visual C++, библиотека классов MFC (Microsoft Foundation Classes), библиотека ATL

(Active Template Library) и собственно среда разработки Windows-приложений MFC.

Ниже приведены примеры работы приложения для разнообразных типов текстур.

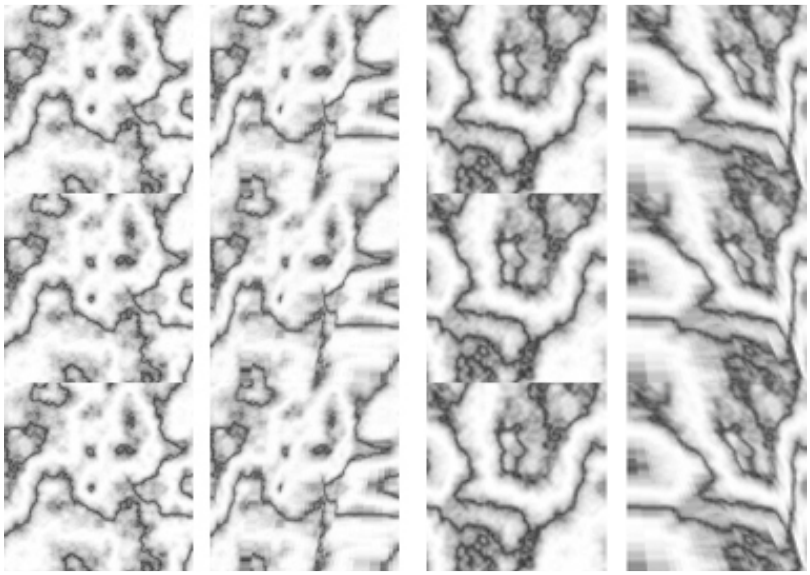

Рис. 5.3. Оригиналы и результаты. Фрагменты мрамора

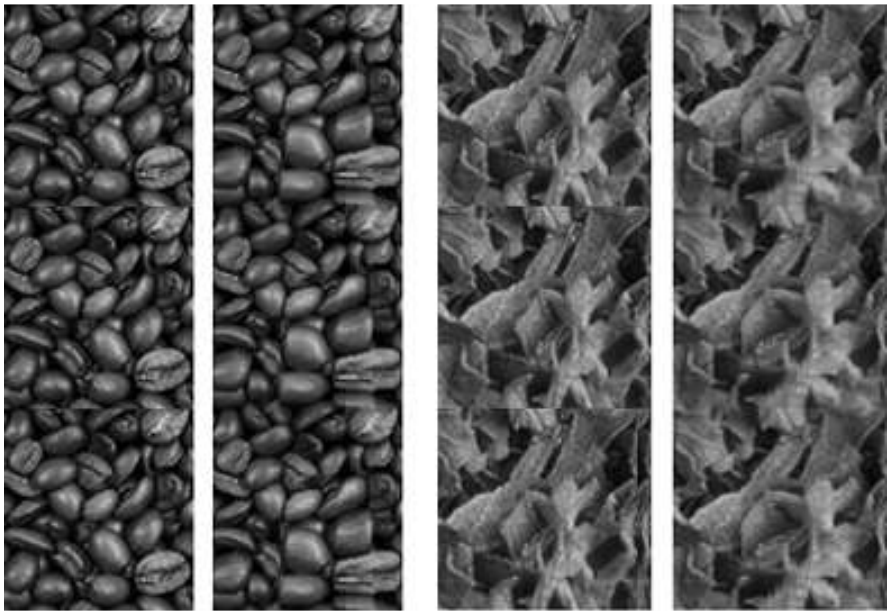

Рис. 5.4. Оригиналы и результаты. Фрагменты кофе и мяты

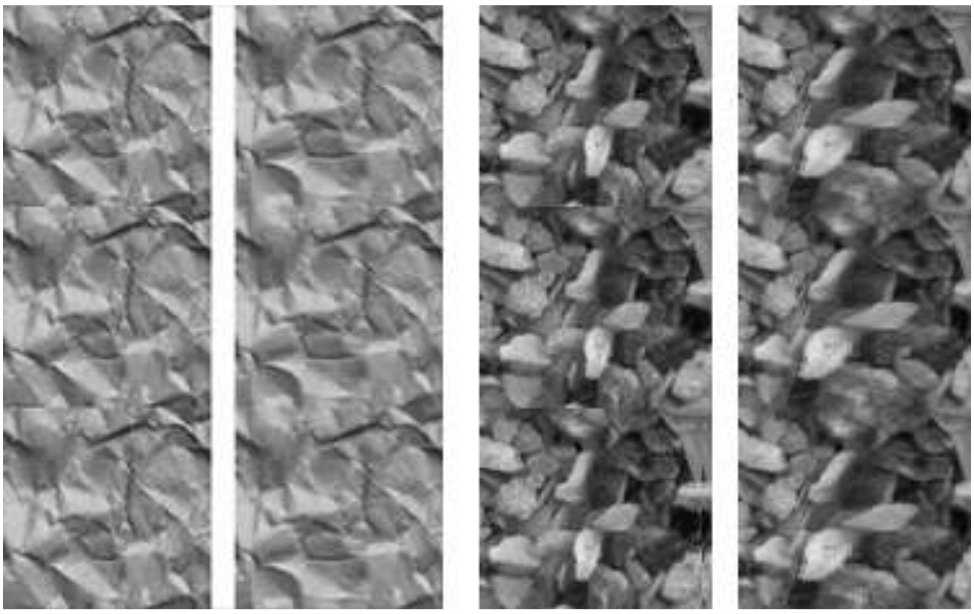

Рис. 5.5. Оригиналы и результаты. Фрагменты бумаги и ракушек

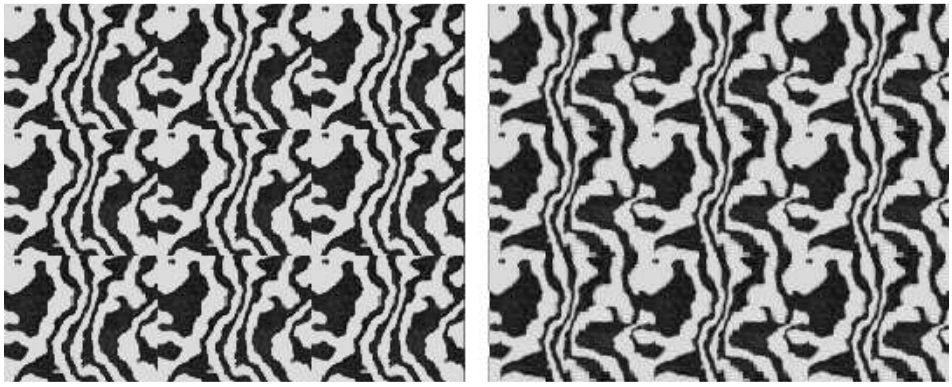

Рис. 5.6. Оригинал и результат. Фрагмент шкуры.
Удаление швов в обоих направлениях

6. Заключение

В первой части работы решалась задача реализации приложения для создания мозаик. На данном этапе поддерживаются следующие симплексы: параллелограммы (и их производные – квадраты, прямоугольники) и правильные шестиугольники. В дальнейшем развитии проекта возможно создание широкой библиотеки функций, которые будут выполнять разнообразные преобразования симплексов и комплексов, а именно: передвижение, удаление вершин симплекса (кроме базовых); выбор типа линий, которые соединяют вершины комплекса (кривые Безье и др.) и др.

Выполненная часть работы показывает, что проект можно совершенствовать и развивать в разнообразных направлениях. Программа может заинтересовать графиков и дизайнеров, как удобный инструмент при разработке оформления интерьеров, обоев, паркетов и т.д.

Во второй части работы исследовался вопрос о создании непрерывных текстур на основе произвольных растровых изображений. Была сделана математическая формализация данной задачи и её возможного решения. Для программной реализации была разработана модель динамического программирования, согласующаяся с предложенными идеями.

В дальнейшем возможно проведение усовершенствования и оптимизации алгоритма с целью ускорения качества и работы программы. Одним из путей является внесение ограничений на возможную ширину пикселя, с целью сокращения количества перебираемых вариантов. Необходимо проверить корректность таких ограничений экспериментально и аналитически. Также следует проанализировать возможность введения в минимизируемый функционал всех трёх компонент цвета. В данной реализации оптимизируется лишь один из каналов цвета.

Разработанное приложение показывает применимость предложенного алгоритма и результаты хорошего качества на многих практических примерах. Наилучшие результаты достигаются на изображениях с небольшим количеством цветов (напр. мрамор и др.). Планируется дальнейшая работа над поиском аналитического решения задачи, а именно подбор функционала, который мог бы дать адекватное по времени выполнения решение хорошего качества.

ЛИТЕРАТУРА

1. Колмогоров А.Н. Паркеты из правильных многоугольников // Научно-популярный физ.-мат. журнал "Квант" 1970. – №3. – С. 3 – 7.
2. Герчук Ю.Я. Что такое орнамент? Структура и смысл орнаментального образа. – М.: Галарт, 1998. – 328 с.
3. Bool F.H., Kist J.R., Losher J.L. M.C. Escher. His Life and Complete Grafic Work. – New York: Abradale Press, 1992. – 349 p.
4. Schattschneider D. M.C. Escher’s classification system for his colored periodic drawings // International Congress on M.C. Escher – 1985. – pp 82 – 96.
5. Cohen M.F., Shade J., Hiller S. Wang tiles for image and texture generation // ACM Trans Graph 22 – 2003. – pp. 287 – 294.
6. Ng T.Y., Wen C., Tan T.S., Generating an ω-tile set for texture synthesis. // Proceedings of computer graphics international 2005 (CGI’05), Stone Brook, New York. – pp. 177 – 184.
7. Efros A.A., Freeman W.T. Image quilting for texture synthesis and transfer // SIGGRAPH – 2001. – pp. 341 – 346.
8. Сеніна А.В. Полігональні регулярні мозаїки на площині // Десята Всеукраїнська (П’ята міжнародна) студентська наукова конференція з прикладної математики та інформатики, Львів ЛНУ – 2007. – С. 151 – 153.
9. Сеніна А.В. Нелінійні плоскі трансформаційні перетворення з функціями переходів // Одинадцята Всеукраїнська (Шоста міжнародна) студентська наукова конференція з прикладної математики та інформатики, Львів ЛНУ – 2008. – С. 226 – 227.
10. Беллман Р. Динамическое программирование, пер. с англ. – М.: Иностранная литература, 1960. – 400 с.

11. Mathematical Art of M.C.Escher, URL: http://www.mathacademy.com/pr/minitext/escher/
12. The official M.C. Escher Website, URL: http://mcescher.com/
13. TextureWorkshop 1.5, URL: http://terminalstudio.com/texture.shtml

Надійшла 26.06.2009.