УДК 515.142.33

Ю.Л. Костюк, Ю.Л. Новиков

ГРАФОВЫЕ МОДЕЛИ ЦВЕТНЫХ РАСТРОВЫХ ИЗОБРАЖЕНИЙ ВЫСОКОГО РАЗРЕШЕНИЯ

Рассматривается универсальная векторная модель многоцветных растровых изображений. Предлагаются эффективные алгоритмы ее построения, основанные на триангулированном представлении векторной модели растра. Рассматриваются аспекты практического применения предложенных алгоритмов.

Решение задачи векторизации многоцветного картографического изображения может быть выполнено двумя путями. Первый– применение специальных аппаратно-программных средств для получения набора бинарных растров, каждый из которых получен выделением из исходного многоцветного растра только тех пикселей, которые имеют заданный или близкий к заданному цвет. Затем происходит последовательная векторизация каждого из полученных бинарных растров, например с применением технологий, описанных в [1, 2]. Главным недостатком такого решения является независимость друг от друга полученных векторных слоев, из-за чего возникает необходимость последующего совмещения полученных наборов объектов на одном листе электронной карты.

Второй путь решения задачи векторизации – построение комплексной векторной модели исходного многоцветного изображения, состоящей из топологически связанных векторных моделей объектов разных цветов. Для реализации данного решения авторами предложена модель *n*-цветного растра и алгоритмы ее построения, изложенные в [3].

В данной работе предлагается первичная модель многоцветного растра и алгоритмы ее построения. Из этой модели возможно эффективное формирование комплексной векторной модели.

Предобработка растра

Пусть имеется многоцветный растр F, который можно представить матрицей $F=\{f_{ij}\}_{M\times N}$, где $f_{ij}=(r,g,b)$ – трехкомпонентный вектор, в котором компоненты r, g, b представляют собой значения интенсивностей красного, зеленого и синего цветов в точке растра (i,j) соответственно, заданные целыми числами в диапазоне от 0 до I. Как правило, этот формат используется при получении сканированных изображений цветных карт. При сканировании на получаемый растр попадает шумовая составляющая – случайное изменение значений интенсивностей цветовых составляющих во многих точках растра.

Для построения векторной модели такого растра необходимо привести его к виду $G=\{g_{ij}\}_{M\times N}$, где g_{ij} – целое число в диапазоне от 0 до B, представляющее собой индекс – номер цвета в палитре базовых цветов $P=\{(r_i^b, g_i^b, b_i^b)\}, i=\overline{1,B}$. Растр G, с одной стороны, не должен содержать шумовой составляющей, т.е. большинство значимых для пользователя объектов растра F должны присутствовать на растре G в виде связных областей одного цвета без «шума». С другой стороны, растр G должен с необходимой для пользователя точностью (без «размытия границ») моделировать объекты на исходном растре.

Решение задачи в общем случае является достаточно трудной задачей, однако в некоторых случаях, когда можно «пожертвовать» небольшими отклонениями получаемых векторных моделей (на величину одного пикселя) в пользу очистки изображения от шума, может быть применен следующий трехэтапный подход.

На первом этапе к изображению F применяется обычный фильтр усреднения пикселей в 8-окрестности или больших окрестностях, с получением растра F'. Растр F' имеет меньшие амплитуды шумов на изображениях объектов, и, кроме того, более размытые, чем на исходном растре, границы областей.

На втором этапе к изображению F' применяется фильтр «постеризации», задача которого – уменьшение количества цветов на растре до заданного (как правило, небольшого) числа. В общем виде это выполняется так. Задается максимальное количество цветов (обозначим его $n_{\max}$), которое необходимо получить, на основе этого максимума рассчитывается число диапазонов для каждой цветовой компоненты, так что произведение этих чисел дает $n_{\max}$. В простейшем случае числа диапазонов для цветовых компонент можно положить равными корню третьей степени из $n_{\max}$. Далее весь ряд значений каждой цветовой компоненты разбивается на полученное число диапазонов этой компоненты одинакового размера.

Затем формируется растр F'', каждый пиксель которого получен из соответствующего пикселя растра F' применением значений каждой из цветовых компонент к среднему значению по тому диапазону данной компоненты, в который попало данное значение. В результате растр F'' будет содержать пиксели не более $n_{\max}$ различных цветов.

На последнем, третьем этапе, выполняем формирование палитры P растра F'' – пронумерованного набора цветов, встречающихся на растре F'', – и формируем растр G, содержащий индексы (номера) цветов соответствующих пикселей растра F'' в палитре P.

Данный подход, наряду с достоинством – уменьшением шума на получаемом изображении и уменьшением количества используемых цветов, имеет и недостаток – возможность непрогнозируемого сдвига получаемых на растре объектов на величину один-два пикселя. Это происходит потому, что соседние, близкие по значениям цвета объекты на растре в результате фильтра усреднения получают размытую границу, которая на этапе «постеризации» может принять значение одного или другого цвета, после чего произойдет границы объекта смещаются. Поэтому наиболее эффективно применение интерактивной процедуры выбора параметров усреднения и «постеризации» с непосредственным участием пользователя. Вкратце суть данной процедуры состоит в следующем: пользователю предлагается возможность диалогового изменения значений параметров в заданном диапазоне с одновременным отображением результата работы алгоритма при заданных параметрах на небольшом участке растра или на уменьшенной копии растра.

Построение векторной модели цветного растра

Будем считать, что в растре G имеются пиксели, каждому из которых приписан какой-либо из цветов палитры P.

Поставим задачу построения векторной модели такого растра. Требуется построить планарный граф линий (ПГЛ) и снабдить его элементы необходимым атрибутивным описанием так, чтобы иметь возможность по значениям атрибутов выполнять выборку элементов графа и эффективно формировать из нее графические объекты, значимые для пользователя (точечные, линейные и полигональные). Образы формируемых объектов на плоскости должны соответствовать с заданной точностью образам объектов на исходном растре.

Определение. Полигональная графовая модель (ПГМ) задается тройкой $M=(V,R,F)$, где

V – множество точек плоскости, являющихся вершинами ПГЛ;

R – множество ребер, каждое из которых соединяет две вершины из V, причем изображения ребер из R на плоскости не пересекаются, поэтому (V,R) представляет собой планарный граф;

F – множество граней ПГЛ, которые представляют собой многоугольники, образованные ребрами из R таким образом, что любая пара многоугольников либо не пересекается на плоскости (за исключением, быть может, граничных точек), либо один многоугольник находится внутри другого. При этом все множество многоугольников F покрывает всю плоскость исходного растра.

Для полноты описания растра в V включаются четыре точки плоскости, соответствующие четырем угловым точкам прямоугольного растра, а в R – четыре отрезка, соединяющие эти четыре вершины (то есть отрезки, соответствующие границам растра).

Грани ПГМ будем называть также полигонами.

Элементы ПГМ перенумеровываются и снабжаются следующими атрибутами:

– вершина графа имеет список всех смежных ей ребер;

– ребро графа имеет номера двух граней, расположенных по обе стороны от него;

– грань задается ребрами, образующими его границу, и имеет атрибут – номер цвета в палитре цветов растра G, область которого покрывается данной гранью, кроме того, грань имеет список номеров граней, расположенных на плоскости внутри нее. При этом, если грань a содержит грань b, а грань b – c, то номер c включается только в список b, образуя, таким образом, «дерево вложенности» граней.

Для построения векторной модели растра предлагается описываемый далее алгоритм. Вначале формируется растр B с изображением границ связных областей одного цвета. Элементы растра $B=\{b_{ij}\}$, который имеет размеры $(M+1)\times(N+1)$, можно представить располагающимися в узлах прямоугольной сетки, образованной границами между пикселями исходного растра, как показано на рис. 1.

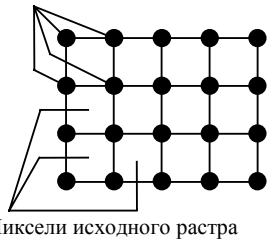

Рис. 1. Исходный растр и растр границ

Элементы растра B должны содержать пять признаков, каждый признак может принимать одно из двух значений (присутствует или нет соответствующая характеристика в данной точке растра B). Эти признаки таковы: b – признак границы; v – признак узловой точки ПГМ; t – признак того, что граница в данной точке уже просмотрена алгоритмом; c – признак узловой граничной точки; d – признак удаленной вершины. Поэтому при программной реализации элементы растра B удобно представлять в виде массива битов, например байта.

При работе алгоритма некоторым пикселям растра G будет приписано нулевое значение (несуществующий цвет в палитре цветов растра G).

Для описания алгоритма удобно сделать следующее определение.

Определение. Назовем граничной линией на растре B такую ломаную линию, образованную точками растра, что соседние узлы этой ломаной являются 4-соседями на растре B, и каждый отрезок ломаной проходит между пикселями разных цветов на растре G при условии, что более длинной ломаной, включающей в себя все точки исходной ломаной, построить нельзя (т.е. граничная линия является максимальной по включению). Пример граничной линии приведен на рис. 2.

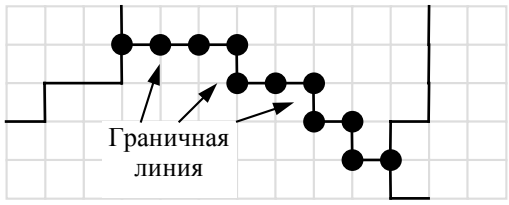

Рис. 2. Граничная линия

Алгоритм построения ПГМ заданного растра.

Исходный растр: $G=\{g_{ij}\}_{M\times N}$.

Параметр δ – «точность» получаемой модели по отношению к исходному растру. Задается в диапазоне $\delta\geq 0$, рекомендуемый из практики диапазон $0\leq\delta\leq 4$. Задается максимум отклонения полученной векторной линии от границы объекта на растре. Масштаб по отношению к размерам пикселей растра $\delta=1$ означает отклонение на размер одного пикселя.

Шаг 1. Формируем растр B с изображением границ связных областей одного цвета.

1.1. Последовательно просматриваются все точки растра G, и для каждой точки $a_4=g_{ij}$ анализируются три соседних пикселя: $a_1=g_{i-1,j-1}$, $a_2=g_{i-1,j}$, $a_3=g_{i,j-1}$ (для точек растра G, имеющих одну или обе координаты, равные 1, величины g_1, g_2, g_3 соответственно полагаются равными индексу несуществующего на растре цвета).

Вычисляем n как число верных неравенств из набора: $g_1\neq g_2$, $g_1\neq g_3$, $g_3\neq g_4$, $g_2\neq g_4$. n может принимать значения 0, 2, 3 или 4.

Если $n=0$, то все признаки элемента b_{ij} устанавливаются равными 0. Если $n=2$ (это содержательно означает, что элемент b_{ij} находится на границе между двумя областями разных цветов на растре G), то признак b элемента b_{ij} устанавливается в 1, все остальные сбрасываются в 0. Если $n>2$, то содержательно это означает, что элемент b_{ij} находится в точке, в которой граничат три или четыре области разных цветов, и тогда признаки b и c элемента b_{ij} устанавливаются в 1, остальные сбрасываются.

1.2. Последовательно просматриваются точки g_{Mj}. Для каждой из них анализируется соседний слева пиксель. Если g_{Mj}=g_{Mj-1}, то признак *c* элемента $b_{M+1,j}$ устанавливается, иначе – он сбрасывается. Признак *b* для всех $b_{M+1,j}$ устанавливается, а *v*, *t* и *d* сбрасываются.

1.3. Аналогично просматриваются точки $g_{i,N}$. Для каждой из них анализируется соседний сверху ($g_{i-1,N}$) пиксель. Если их значения равны, то признак *c* элемента $b_{i,N+1}$ устанавливается, иначе – сбрасывается. Также для всех $b_{i,N+1}$ признак *b* устанавливается, а *t*, *v* и *d* – сбрасываются.

Шаг 2. Формирование начального набора узловых точек.

Создается пустое множество вершин *V*.

Последовательно просматриваются все пиксели растра *B,* и если признак *c* пикселя b_{ij} установлен, то в *V* добавляется вершина с координатами (*i*,*j*). При этом признак *v* пикселя b_{ij} устанавливается.

Шаг 3. Построение граней, ребер и формирование избыточного набора вершин.

3.1. Создается копия начального множества вершин, V_H=*V*.

3.2. Для каждой вершины *w* из V_H выполняются в цикле шаги 3.2.1 – 3.2.3.

3.2.1. Находим на растре *B* пиксель b_{ij} , где (*i*,*j*) – координаты *w*. Этот пиксель соответствует точке, в которой на растре *G* соседствуют три или четыре цвета, например, как показано на рис. 3.

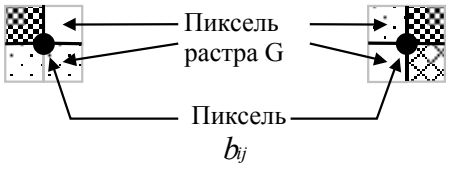

Рис. 3. Пиксель на растре *G*,
значение которого проверяется

3.2.2. В цикле просматриваем 4-соседей точки (*i*,*j*) на растре *B* в следующем порядке: снизу, справа, сверху, слева. (в направлении против часовой стрелки), пока не найдем пиксель b_{pq} с установленным признаком *b*. Проверим, какое значение имеет пиксель растра *G*, расположенный между пикселями b_{ij} и b_{pq} и справа от отрезка (*i*,*j*)–(*p*,*q*), как показано на рис. 4.

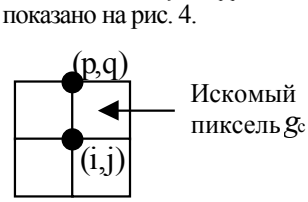

Рис. 4. Пиксель на растре *G*,
значение которого проверяется

Если его значение не равно 0, то сохраняем значение этого пикселя в переменной C_I и выходим из цикла. Иначе – переходим к следующему шагу цикла по пикселям b_{pq}

3.2.3. Восстанавливаем границу области одного цвета растра *G*, которой принадлежит пиксель $g_{\copyright}$. Для этого вызываем алгоритм построения границы области растра *G* (описываемый ниже) с параметрами: текущая точка – (*i*,*j*), следующая точка –(*p*,*q*).

3.3. Завершение обхода области на растре *G*. На этом шаге создаем новую грань *F*, задавая ее набором ребер R_F и индексом цвета C_I.

Шаг 4. Построение векторных моделей изолированных замкнутых граничных линий.

Просмотрим последовательно пиксели растра *B* в поисках пикселя с установленным признаком *b* и сброшенным *t*. Создаем вершину в этой точке и добавляем ее в множество V_H. Далее выполняем все действия по обходу граничной линии, описанные на шаге 3, начиная с текущей точки. После выполнения этих действий возвращаемся к поиску пикселя с установленным *b* и сброшенным *t*.

Шаг 5. Уменьшение количества вершин ПГМ (генерализация). К полученной модели растра применяем алгоритм генерализации полной ПГМ, описанный ниже.

Конец алгоритма построения ПГМ.

Сделаем некоторые пояснения к алгоритму.

На шаге 3 строится большинство ребер ПГМ так, что на плоскости эти ребра будут точно соответствовать границам между связными областями точек растра одного цвета. При этом полученные ребра не пересекают такие связные области.

Как нетрудно заметить, на шаге 3 будут построены векторные модели только тех граничных линий, которые имеют точки пересечения с другими граничными линиями. Для построения векторных моделей оставшихся изолированных граничных линий (которые будут замкнутыми) предназначен шаг 4, на котором выполняются те же действия, что и на шаге 3, но для еще не векторизованных контуров.

Результат работы шагов 3 и 4 представляет собой полную векторную модель граничных линий и областей, которая, однако, слишком избыточна. Для уменьшения этой избыточности могут быть предложены самые разнообразные алгоритмы, причем выбор конкретного алгоритма напрямую зависит от специфики решаемых при векторизации задач.

Рассмотрим алгоритм генерализации, основанный на подходе, описанном в [4].

Определим объекты, которые необходимо подвергнуть генерализации. Рассмотрим множество вершин ПГМ *V*. Вершины, которым инцидентны три или четыре ребра ПГМ, не следует удалять из ПГМ, так как они несут существенную информацию – точки стыковки областей трех или четырех различных цветов. Вершины, которым инцидентны только два ребра, могут быть удалены, если получаемые ребра отклоняются от исходной граничной линии не более чем на δ. При этом уменьшается как количество вершин, так и ребер. Количество граней ПГМ остается неизменным. Таким образом, задача предлагаемого алгоритма – удаление из ПГМ вершин с валентностью два, которое не приводит к отклонению получаемых ребер от исходной границы на растре на величину более δ.

После завершения шага 5 алгоритма построения ПГМ у нас имеется генерализованная ПГМ, заданная множествами вершин *V*, ребер *R* и множеством граней *F*.

Алгоритм генерализации полной ПГМ.

Перебираем в цикле вершины из множества V_H и выполняем для них шаг 1.

Шаг 1. Пусть выбранная из V_H вершина – w_1. Просматриваем все ребра, инцидентные w_0 (их может быть три или четыре), и в цикле выполняем для них шаг 2.

Шаг 2. Имеем вершину w_1 и ребро, инцидентное ей, которое обозначим r_1. Сформируем временные списки

вершин (V_t) и проходящих между ними ребер (R_t) следующим образом. Начиная от вершины w_1 и ребра r_1, находим противоположную текущей вершину, инцидентную текущему ребру, и заносим ее во временный список V_t. В список R_t заносим текущее ребро. Если все ребра, инцидентные текущей вершине, имеют длину больше одного пикселя, то помечаем данную вершину как неудаляемую. Если новая вершина имеет валентность 3 или более, то выходим из цикла, иначе – находим ребро, инцидентное новой вершине, но не совпадающее с текущим. Делаем новую вершину и найденное ребро текущими и повторяем шаг цикла.

Шаг 3. Находим первую неудаляемую вершину в списке V_t, которой необходимо является вершина w_1. Просматриваем список V_t и находим следующую неудаляемую вершину. Обозначим ее w_2. Такая вершина всегда найдется, по крайней мере, неудаляемой является последняя вершина списка V_t. Все вершины и ребра, находящиеся между первой и второй неудаляемыми вершинами, заносим в порядке их следования в список $W_©$. Переходим к следующему шагу с параметрами $w_{(H)}=w_1$, $w_{(K)}=w_2$, $W=W_©$.

Шаг 4. Генерализация ломаной, заданной неудаляемыми концевыми точками $w_{(H)}$ и $w_{(K)}$ и набором промежуточных точек W.

4.1. Среди набора W выбираем точку, наиболее удаленную от отрезка $(w_{(H)},w_{(K)})$, обозначим ее w_D. Пример такой точки приведен на рис. 5.

Если расстояние от w_D до отрезка $(w_{(H)},w_{(K)})$ меньше δ, то переходим к выполнению шага 4.2. Иначе – переходим к выполнению шага 4.3.

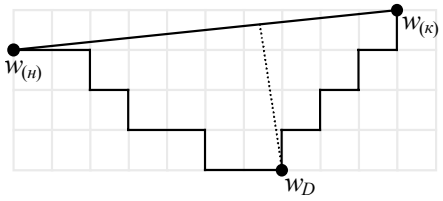

Рис. 5. Граничная линия
и точка с наибольшим отклонением
от отрезка между начальной и конечной точками

4.2. Считаем, что по условию генерализации ломаную можно заменить отрезком$(w_{(H)},w_{(K)})$, поэтому выбираем из ПГМ все вершины из W, а также все ребра, инцидентные W, и заменяем их одним ребром $(w_{(H)},w_{(K)})$. Перед заменой определяем, какие грани находились по обе стороны от ломаной (номера этих граней содержатся в атрибутах любого ребра ломаной, например, соединяющегого$w_{(H)}$ с первой вершиной из W), и приписываем соответствующие значения атрибутам вновь созданного ребра $(w_{(H)},w_{(K)})$. Вносим соответствующие изменения в списки ребер левой и правой граней рассматриваемой ломаной. Завершаем работу алгоритма.

4.3. Объявляем точку w_D неудаляемой и рекурсивно применяем шаг 4 со следующими параметрами: $w_{(H)}$ – та же; параметр $w_{(K)}$) принимает значение w_D; параметр W принимает значения подсписка W $(w_1,w_2,\ldots,w_{k-1})$, где k – индекс w_D в списке W.

Затем рекурсивно применяем шаг 4, используя следующие параметры: вместо $w_{(H)}$ используется w_D; в качестве параметра $w_{(K)}$ используется текущее значение $w_{(K)}$; в качестве W используется подсписок $(w_{k+1}, w_{k+2},.., w_n)$ исходного списка W, где k – индекс w_D в списке W, а n – индекс последней вершины в списке W.

Конец алгоритма.

Приведем описание алгоритма построения границы области растра G.

При работе алгоритма создается очередь Q, предназначенная для хранения максимум трех пикселей растра B. Напомним, очередь представляет собой структуру данных, элементы которой линейно упорядочены, добавление элементов в которую происходит с одного конца, а удаление – с другого.

Алгоритм построения границы области одного цвета растра *G*.

Параметры: текущая точка на растре В – b_{ij}, следующая точка – b_{pq}.

Шаг 1. Создаем пустую очередь Q. Выполняем в цикле шаг 2.

Шаг 2. Очищаем очередь Q и помещаем в нее текущую точку. Проверяем, установлен ли признак t пикселя b_{pq}. Если нет, то выполняем шаг 3. Иначе– выполняем шаг 4.

Шаг 3. Выполняем создание ребер, проходящих вдоль данной граничной линии, в цикле перемещая текущую точку по пикселям растра B, у которых установлен признак b. В точках ветвления и пересечения выбираем самое правое относительно текущего направление. При этом выполняем (в цикле) следующие действия.

3.1. Во всех пикселях растра B, по которым прошел алгоритм, устанавливаем признак t.

3.2. В начальной точке движения устанавливаем, кроме того, признак v.

3.3. В пикселях растра G, расположенных между текущей и предыдущей точкой на растре B, справа по отношению к отрезку от предыдущей к текущей точке, записываем значение 0 (индекс несуществующего цвета).

3.4. Если в текущей точке с координатами (l,r) граничная линия изменяет направление, то создаем вершину ПГМ с координатами (l,r) и помещаем ее в множество V; создаем ребро, начинающееся в предыдущей вершине на этой граничной линии и оканчивающееся во вновь создаваемой вершине. У пикселя b_{lr} устанавливаем признак v.

3.5. Если в текущей точке пиксель b_{lr} имеет установленный атрибут c, то в данной точке существует вершина, принадлежащая множеству V_H; находим ее и создаем ребро, оканчивающееся в точке (l,r).

3.6. Все вновь создаваемые на шагах 3.4, 3.5 ребра заносим во временный список ребер R_F.

3.7. Если признак d пикселя b_{ij} не установлен, то заносим пиксель b_{ij} в очередь Q. Последний элемент очереди удаляется.

3.8. Если элементы очереди Q представляют собой одну из конфигураций, изображенных на рис. 6, то удаляем пиксель q_2 из очереди Q и выполняем коррекцию вершин и ребер следующим образом. Если в V_H существует вершина, имеющая те же координаты, что и q_1 то удаляем из V_H вершину, имеющую координаты q_2, и ребро q_1–q_2 заменяем ребром q_1–q_3. Иначе (q_1 не является вершиной) изменяем координаты вершины q_2 так, чтобы они стали равными q_1. Признак d пикселя растра B, имеющий координаты q_2, устанавливается.

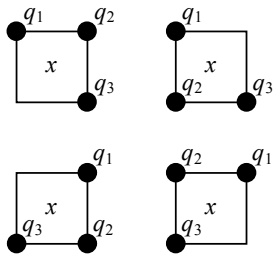

Рис. 6. Изменяемые конфигурации точек на граничной линии: q_1, q_2, q_3 – элементы очереди Q. Текущая точка совпадает с q_3; x – пиксель растра G.

3.9. Если текущая точка – начальная точка движения, то создаем последнее ребро, оканчивающееся в начальной вершине, выходим из цикла и завершаем работу алгоритма.

3.10. Как только текущий пиксель b_{lr} имеет установленный признак t, создаем вершину в точке (l,r), соответствующее ребро и выходим из цикла.

Шаг 4. Текущая граничная линия уже создана, о чем свидетельствует признак t, поэтому просто отслеживаем ее. Для этого так же, как и на шаге 3 в цикле, двигаемся вдоль граничной линии, выполняя следующие действия.

4.1. В начальной точке движения, которая обязательно соответствует вершине из $V_{\text{Н}}$, находим ребро, идущее в том же направлении, что и рассмотренная граничная линия. Заносим это ребро в список R_F и сразу перемещаемся в другую вершину этого ребра.

4.2. В текущей точке, которая обязательно соответствует вершине, принадлежащей либо $V_{\text{Н}}$, либо V, выбираем граничную линию, направление которой – самое правое по отношению к текущему направлению. Если следующая на этом направлении точка имеет установленный признак t, то так же, как и в случае 4.1, выбираем ребро, идущее в том же направлении, что и рассматриваемая граничная линия.

Заносим это ребро в список R_F и сразу перемещаемся в другую вершину этого ребра. Если же следующая на этом направлении точка не имеет установленного t, то переходим к выполнению шага 1, не перемещаясь в новую точку.

4.3. Если текущая точка – начальная точка движения, то выходим из цикла и завершаем работу алгоритма.

4.4. При движении вдоль ребер, найденных на шагах 4.1 и 4.2, в пиксели растра G, находящиеся справа от ребер, записываем 0 (индекс несуществующего цвета).

Конец алгоритма.

Триангулированное представление ПГМ

Выше был описан алгоритм построения ПГМ, результат работы которого – наборы вершин, ребер и граней – при практической реализации представляются в виде простых структур данных – списков или массивов.

Однако при частых операциях выборки элементов ПГМ по заданным координатам на плоскости поиск подходящих объектов в неупорядоченных списках становится неэффективен. Рассмотрим триангулированное представление ПГМ, позволяющее эффективно реализовать операции выборки и регионального поиска элементов ПГМ, что имеет важное значение при объектной векторизации.

Определим триангулированное представление ПГМ как триангуляцию T, построенную на множестве точек плоскости – вершин ПГМ – V таким образом, что все ребра из множества ребер ПГМ R входят в триангуляцию T в качестве ее ребер, а грани ПГМ представляют собой множества смежных треугольников триангуляции.

Построенная таким образом триангуляция представляет собой структуру $T=(V,R\cup R_h,\Theta)$, где V – исходное множество вершин; R – множество ребер – ограничений триангуляции; R_h – множество ребер, построенных в процессе триангуляции, Θ – множество треугольников, заданных вершинами и ребрами, их образующими. Ребра из R_h будем называть «невидимыми ребрами», так как они являются внутренними для процесса построения ПГМ и невидим для пользователя.

При работе алгоритма будет сформировано дерево вложенности граней, введенное в разделе 2. Оно образуется гранями, когда каждая имеет список номеров граней, находящихся внутри данной. В процессе работы алгоритма построения дерева вложенности на ребра множества R будут наноситься пометки (признак того, что данное ребро просмотрено).

Алгоритм построения триангулированного представления ПГМ.

Шаг 1. Построение триангуляции Делоне с ограничениями. На множестве точек плоскости V построим триангуляцию Делоне T с ограничениями, в качестве которых выступают ребра R.

Шаг 2. Разметка треугольников триангуляции по принадлежности к граням. Сделаем это следующим образом. Просматриваем последовательно все ребра множества R, выбираем номера левой и правой граней каждого ребра и записываем эти номера в атрибут – «номер грани-владельца» правому и левому треугольнику ребра соответственно.

Шаг 3. Построение «дерева вложенности» граней, введенного в разделе 2.

3.1. Просматриваем в цикле список граней ПГМ. Для каждой грани выполняем шаг 3.2.

3.2. Выбираем еще не помеченное ребро из списка ребер текущей грани. Данное ребро, если такое существует, необходимо принадлежит либо внешнему контуру текущей грани, либо одному из внутренних. Если такого ребра не существует, то выходим из цикла. Обходим последовательно все ребра данной грани так, чтобы треугольники, образующие грань, всегда оставались справа по отношению к направлению обхода. При обхода контура подсчитываем сумму углов поворотов, сделанных в вершинах ПГМ. В результате получаем обход внутреннего или внешнего контура, как показано на рис. 7. В процессе обхода помечаем все ребра как просмотренные.

Если полученная сумма углов положительна, то контур – внутренний, и для него выполняем шаг 3.3.

3.3. Повторно обходим контур и заносим в список внутренних граней номера всех граней, находя-

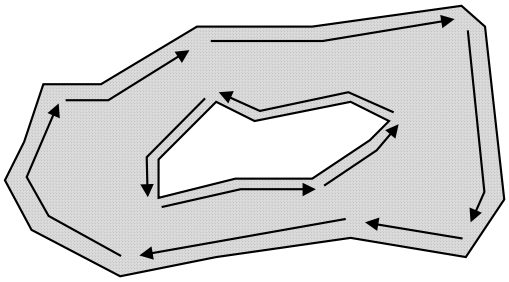

Рис. 7. Обход внешнего и внутреннего контуров грани

дящихся слева от ребер контура. Номера граней, находящихся слева от контура, определяются из атрибута – «номера грани» – треугольников, смежных слева ребрам контура.

Шаг 4. Дерево вложенности построено. Каждая грань имеет список номеров граней, содержащихся строго внутри нее.

Конец алгоритма.

Сделаем несколько замечаний к алгоритму.

Для построения триангуляции Делоне с ограничениями следует использовать эффективные алгоритмы ее построения, использующие кэширование, предложенные в [5]. В [6] представлена модификация данных алгоритмов, эффективно реализующая построение триангуляции с использованием целочисленной арифметики. Трудоемкость этих алгоритмов в среднем составляет $O(n)$, где n – число точек множества V.

Построенная на шаге 1 триангуляция такова, что каждый из треугольников полностью находится (за исключением, может быть, границы), внутри какой-либо грани ПГМ. Поэтому следует приписать каждому треугольнику атрибут – номер грани, содержащей его.

После выполнения алгоритма не требуется хранить в атрибутах граней списки ребер ПГМ, их образующих, так как многоугольник, образуемый любой гранью, может быть получен объединением треугольников триангуляции, имеющих одинаковое значение атрибута – номера грани.

Отметим, что наиболее трудоемкий этап этого алгоритма – первый шаг, на котором выполняется триангуляция. Трудоемкость его составляет $O(n)$ в среднем, где n – число вершин ПГМ. Трудоемкость шагов 2 и 3 составляет $O(m)$ в худшем, где m – число ребер ПГМ, что имеет тот же порядок, что и $O(n)$. Данные оценки трудоемкости получены на основе [5].

Полученное триангулированное представление ПГМ имеет ключевое значение для реализации эффективной объектной векторизации, в процессе которой выполняются массовые операции поиска объектов ПГМ (вершин, ребер и граней) по следующим критериям: поиск объекта, ближайшего к заданной точке плоскости, поиск объекта, попавшего в заданную на плоскости область, поиск ближайшего к заданному объекта того же типа. Приведем основные приемы реализации эффективного поиска объектов ПГМ.

Поиск вершины ПГМ, ближайшей к указанной точке. По кэш-таблице треугольников, построенной при триангулировании, определяется треуго-льник, находящийся на удалении от заданной точки не большем, чем размер ячейки кэш-таблицы. Затем по ребрам треугольника за константное в среднем время находится треугольник, содержащий заданную точку, и определяется ближайшая к ней вершина ПГМ.

Поиск ребра ПГМ, ближайшего к указанной точке, выполняется аналогично.

Поиск грани ПГМ, содержащей указанную точку. Аналогично поиску вершины ПГМ по кэш-таблице находится треугольник, содержащий данную точку. Результат определяется по значению атрибута – номера грани – «владельца» данного треугольника.

Поиск вершины ПГМ, ближайшей к заданной вершине, находящейся в заданном направлении. Выбираем все ребра, выходящие из заданной вершины. Выбираем из них ребро, имеющее наименьшее отклонение от заданного направления. Вершина, смежная данной по выбранному ребру, является искомой вершиной.

Объектная векторизация

Как видно из вышеизложенного, построение ПГМ цветного растра выполняется автоматически. Однако построение графических объектов, значимых для пользователя, невозможно без его непосредственного участия. Применение ПГМ позволяет частично автоматизировать этот процесс. Объектной векторизацией будем называть процесс построения векторных объектов в полуавтоматическом режиме, когда пользователь указывает на растре одну-две точки, а алгоритм векторизации восстанавливает весь объект, используя ПГМ для анализа.

Рассмотрим некоторые примеры алгоритмов полуавтоматической векторизации.

Автоматическое построение точечного объекта, имеющего изображение на растре в виде небольшой «кляксы», по указанной пользователем точке. Более точно задача может быть сформулирована так. Пусть на растре имеется изображение точечного объекта. Пользователь указывает точку на плоскости, находящуюся внутри этого объекта или недалеко от него. Требуется найти точку на плоскости, которая находится в середине данного объекта.

С применением ПГМ решение данной задачи становится очевидным и выглядит так. Находим треугольник триангуляции и грань ПГМ, в которые попала указанная точка. Определяем, является ли эта грань выпуклой и имеющей небольшой размер (проверка размера может выполняться для треугольников грани, последовательно просматриваемых алгоритмом). Если данное условие не выполняется, то переходим к рассмотрению ближайшей соседней грани.

Иначе (условие на выпуклость и ограничение размера выполняется) – находится центр грани как точка, координаты которой являются средним арифметическим от соответствующих координат точек грани. Данный центр будет результатом решения задачи.

Автоматическое построение линейного объекта по ПГМ растра и указываемым пользователем одной или нескольким точкам. Поставим задачу следующим образом. Пусть на растре имеется изображение линейного объекта одного цвета. Пользователь указывает одну-две точки на этом объекте. Необходимо построить ломаную или набор ломаных, аппроксимирующих данный линейный объект по его

средней линии. Пример аппроксимирующих ломаных приведен на рис. 8.

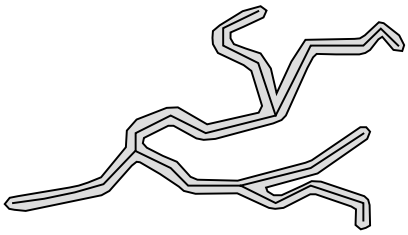

Рис. 8. Ломаные, аппроксимирующие линейный объект

Для решения этой задачи можно предложить такой способ.

Так как имеется ПГМ растра, то легко можно установить грань, в которую попадает указанная пользователем точка (или выбрать из нескольких граней, если пользователь указывает несколько точек). Эта грань является векторным объектом линейного объекта на растре, однако грань является площадным объектом, что вызывает необходимость построения «скелета» данной грани. Полученный «скелет» и будет являться результатом данного алгоритма объектной векторизации.

Рассмотрим предлагаемый алгоритм построения «скелета» протяженной грани ПГМ. Введем несколько определений.

Определение. Назовем протяженной гранью ПГМ – грань F ПГМ, из каждой точки которой можно провести по крайней мере один отрезок до границы F, состоящий только из внутренних точек F, длина которого меньше некоторого фиксированного δ, и, кроме того, существуют по крайней мере две точки, принадлежащие F, такие, что предел минимума расстояния между ними, измеренного по кривой, проходящей только по внутренним точкам F, намного больше δ.

Определение. Назовем каркасом грани F ПГМ множество точек плоскости, расстояние от каждой из которых до по крайней мере двух различных граничных точек F одинаково.

Определение. Назовем «скелетом» грани ПГМ такую ломаную линию (набор ломаных), что каждая из ее вершин находится на одинаковых расстояниях от двух граничных точек грани.

Определение каркаса, предложенное в [4], является конструктивным, но трудоемкость его построения слишком высока. Поэтому введено определение скелета, который будет достаточно точно имитировать каркас, обладая в то же время приемлемой трудоемкостью построения.

Алгоритм построения скелета протяженной грани ПГМ.

Обозначим точку, указанную пользователем, символом x. В процессе работы алгоритма некоторые ребра триангуляции помечаются как просмотренные. После работы алгоритма все пометки на ребрах удаляются.

Шаг 1. Определяем треугольник триангуляции, в который попала точка x. Назовем его t.

Шаг 2. Запускаем процедуру формирования ломаных с параметром – треугольником t (шаг 3).

Шаг 3. Формирование ломаных, начиная от заданного треугольника t_0.

3.1. Находим середины непомеченных «невидимых» ребер (ребер, принадлежащих множеству R_h триангуляции) данного треугольника. Если таких ребер нет, то выходим из рекурсии. Добавляем полученный отрезок между серединами ребер к ломаной-результату.

3.2. Для каждого из «невидимых» ребер находим треугольник, смежный с данным, имеющий с данным общее ребро. Рекурсивно выполняем для него шаг 3.2.

Шаг 4. Генерализация полученного объекта. Полученный на шаге 3 объект имеет вид планарного графа, который может иметь значительное количество несущественных для пользователя деталей, представленных на рис. 9.

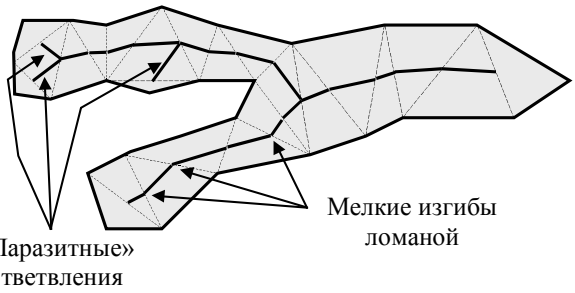

Рис. 9. Несущественные для пользователя детали объекта

Для избавления от них применяется данный шаг.

4.1. Формирование списка конечных ребер дерева. В список R_f помещаются все ребра, имеющие одну концевую вершину (вершину, инцидентную только одному ребру) и другую вершину, находящуюся на расстоянии не более gamma от ближайшей (по ребрам ломаной) узловой вершины (вершины, имеющей более чем одно инцидентное ребро). Если список R_f пуст, то выходим из цикла, переходя на шаг 4.3.

4.2. Просматриваем список R_f и удаляем из него все ребра, имеющие длину не более фиксированного δ. Возвращаемся на шаг 4.1.

4.3. Полученная на предыдущих шагах алгоритма ломаная будет избавлена от мелких ответвлений, однако будет содержать слишком много промежуточных точек. Для уменьшения их количества применяем генерализацию полученных ломаных, как описано на шаге 4 алгоритма построения ПГМ растра.

Конец алгоритма.

Можно показать, алгоритм имеет линейную трудоемкость относительно числа «невидимых» и «видимых» ребер триангуляции, попавших внутрь данной грани, что достаточно эффективно для данной задачи.

Полуавтоматическое построение площадного объекта по ПГМ растра и указываемым пользователем одной или нескольким точкам. Данная задача имеет следующую общую постановку. Пусть на растре имеется одно или несколько фрагментов изображения одного, значимого для пользователя площадного объекта одного цвета. Пользователь указывает одну или более точек так, чтобы в каждый фрагмент объекта попала хотя бы одна точка. Необходимо построить многоугольник, аппроксимирующий заданный объект по его контуру.

Заметим, что потребность создания одного объекта по нескольким фрагментам на растре возникает в случаях, когда поверх площадного объекта на исходном изображении изображены линейные объек-

ты других типов, либо на площадном объекте имеются надписи, значки и т.п. атрибутивная информация. Кроме того, если площадной объект на исходном изображении ограничен линейным, то возникает задача точного определения границы объекта: в зависимости от решения пользователя границей площадного объекта может считаться граничная линия между линейным и площадным объектами либо осевая линия линейного объекта. Для решения данной задачи с использованием построенной ПГМ может использоваться следующий подход.

Определяются грани ПГМ, в которые попали точки, указанные пользователем. Если таких граней несколько, то последовательно выполняется объединение граней в одну, как описано далее.

Выбираем две из еще не объединенных граней. Находим две граничные вершины ПГМ (одна – на одной грани, вторая – на другой), расстояние между которыми минимально. Находим треугольник (или два треугольника, смежных друг другу), находящийся между найденными вершинами. Помещаем этот треугольник во множество включаемых в объект треугольников, которое обозначим символом I.

Формируем множество включаемых в объект треугольников как множество треугольников, каждый из которых имеет общее ребро с одной из выбранных граней, общую вершину с другой гранью и общее ребро с уже включенными в I треугольниками. Объединение треугольников из I образует многоугольник, имеющий общие части границы с выбранными гранями. Поэтому две грани и I можно логически объединить в одну, получая многоугольник. Этот процесс продолжается итеративно, пока в результате не останется одна логическая грань, которую обозначим символом P.

Если пользователю не требуется создание границ площадных объектов, проходящих по серединам смежных линейных объектов, то на этом процедура построения площадного объекта заканчивается.

Иначе производится построение точных границ полученного площадного объекта P, выполняемое следующим образом.

Строим набор треугольников, смежных друг с другом, принадлежащих одной грани ПГМ и имеющих с границей полученного площадного объекта общее ребро или вершину. Полученный связный набор треугольников обозначим символом S_t. Для полученного набора треугольников S_t, который является многоугольником, строим «скелет» алгоритмом построения скелета протяженной ПГМ (см. выше), который обозначим $L(S_t)$. Если толщина полученного объекта (которая вычисляется как удвоенное расстояние от скелета до граничных линий) меньше заданного порога, то считаем этот набор частью некоторого линейного объекта и заменяем границу площадного объекта P, проходящую между P и S_t, на $L(S_t)$. Эту процедуру применяем последовательно для всех граней, имеющих с объектом P общую границу. В результате граница объекта P будет скорректирована во всех участках, в которых объект P граничит с линейными объектами.

Таким образом, построение площадных объектов целесообразно выполнять в полуавтоматическом – интерактивном режиме, так как конечный результат зависит от решаемой пользователем задачи.

Заключение

Предлагаемая авторами полигональная графовая модель цветного растра является более общей по сравнению с предложенными ранее в работах [7], [2] графовыми моделями бинарных растров, так как позволяет достаточно полно и с необходимой степенью точности моделировать растры, содержащие произвольное количество цветов. Вместе с тем данная модель получает более точные результаты, чем модели, использующие скелетизацию и построение моделей скелетных линий (см., например, [7]). Кроме того, данный алгоритм обладает меньшей трудоемкостью построения модели растра и ориентирован на интерактивную обработку, что повышает эффективность всего процесса построения цифровых моделей растровых изображений.

ЛИТЕРАТУРА

1 . *Костюк Ю.Л., Новиков Ю.Л.* Графовые модели растровых изображений в задаче векторизации // Материалы Международной конференции «Дискретный анализ и исследование операций» (Новосибирск, 26 июня – 1 июля 2000). Новосибирск: Изд-во Ин-та математики, 2000. С. 212.

2 . *Обработка* и отображение информации в растровых графических системах. Минск: ИТК АН БССР, 1989. 180 с.

3 . *Новиков Ю.Л.* Полигонально-линейные графовые модели растровых изображений // Геоинформатика-2000: Труды Международной научно-практической конференции / Под ред. А.И. Рюмкина, Ю.Л. Костюка, А.В. Скворцова. Томск: Изд-во Том. ун-та, 2000. С. 50–55.

4 . *Дуда Р., Харт П.* Распознавание образов и анализ сцен: Пер. с англ. М.: Мир, 1976. 511 с.

5 . *Скворцов А.В., Костюк Ю.Л.* Применение триангуляции для решения задач вычислительной геометрии // Геоинформатика. Теория и практика. Вып. 1. Томск: Изд-во Том. ун-та, 1998. С. 22–47.

6 . *Скворцов А.В.* Особенности реализации алгоритмов построения триангуляции Делоне с ограничениями // Наст. журн.

7 . *Розенфельд А.* Распознавание и обработка изображений с помощью вычислительных машин: Пер. с англ. М.: Мир, 1972. 230 с.

Статья представлена кафедрой теоретических основ информатики факультета информатики Томского государственного университета, поступила в научную редакцию номера 3 декабря 2001 г.