Министерство образования Российской Федерации
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет информатики
Кафедра теоретических основ информатики

УДК 681.03

ДОПУСТИТЬ К ЗАЩИТЕ В ГАК
Зав. кафедрой, доцент, д.т.н.
_____Ю.Л. Костюк
«___»_____2003 г.

Кон Алексей Борисович

ВЕКТОРИЗАЦИЯ МНОГОЦВЕТНЫХ РАСТРОВЫХ ИЗОБРАЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ ТРИАНГУЛЯЦИИ

Дипломная работа

Научный руководитель,
доцент, д.т.н. _____ Ю. Л. Костюк

Исполнитель,
студент гр.1481 _____ А.Б. Кон

Электронная версия дипломной работы помещена
В электронную библиотеку. Файл
Администратор

Томск - 2003

РЕФЕРАТ

Дипломная работа: 53 с., 18 рис., 3 источника, 3 приложения.

МАШИННАЯ ГРАФИКА, ВЕКТОРИЗАЦИЯ, ОЦИФРОВКА, ТРИАНГУЛЯЦИЯ С ОГРАНИЧЕНИЯМИ, ОБРАБОТКА РАСТРА, СКЕЛЕТНЫЕ ЛИНИИ, ADOBE ILLUSTRATOR, PLUGIN

(1) Объект исследования – алгоритмы векторизации полноцветных растров.

(2) Цель работы – исследование задачи векторизации многоцветных растров, разработка и реализация алгоритма, решающего данную задачу.

(3) Метод исследования – экспериментальный (на ЭВМ).

(4) Результат работы – проведен обзор методов векторизации, рассмотрены существующие алгоритмы в свете применения к многоцветным растрам, реализован подход основанный на применении триангуляции в задачах векторизации, предложены и реализованы алгоритм разделения линейных и площадных объектов по триангуляции на основе соотношения площади к длине ребер треугольников и алгоритм первичной аппроксимации граничных линий.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Векторизацией называют процесс получения векторной модели на основе растрового изображения. Суть проблемы состоит в том, что в настоящее время не существует метода, позволяющего полностью автоматизировать перевод в векторную форму информации, представленной в графическом виде. Во многом это связано с тем, что алгоритмически не решена задача однозначной трактовки графических изображений. Под векторной формой далее будем понимать набор объектов, которые задаются точками, ломанными либо многоугольниками.

Однако имеется множество областей, в которых существует необходимость в подобных преобразованиях. В первую очередь это геоинформационные системы, где для создания законченных продуктов необходимо преобразование традиционных источников картографической информации – бумажных носителей – в электронную форму. Среди других сфер применения алгоритмов векторизации можно назвать САПР, дизайн и подготовку печатных/электронных изданий. В каждой из этих областей существуют свои особенности и сложности. Например, в ГИС обычно подразумевается, что исходные растровые изображения карт используют ограниченное и достаточно небольшое число цветов, но при этом необходимо обрабатывать очень большие по размерам изображения. Также в ГИС и системах проектирования часто исходные данные содержат смесь линейных (сплошных, пунктирных, штрих-пунктирных и др.), площадных, символьных объектов, условных знаков, заштрихованных областей и т.д. Что также сильно увеличивает сложность задач распознавания и векторизации. Что касается САПР-приложений, то там существует потребность выделения таких специфичных случаев, как прямые углы между линиями и дуги окружностей.

Задачи, которые существуют в области дизайна и полиграфии, менее всего формализованы и определены. В большинстве случаев, в подобных задачах растровые изображения имеют меньшие размеры, чем в ГИС и САПР, обычно подразумевается, что символьно-текстовая часть изображения не нуждается в распознавании, т.к. легко может быть добавлена вручную (так же необходимо отметить тот факт, что для распознавания текстов разработан целый класс алгоритмов, достаточно сильно отличающихся от алгоритмов векторизации). С другой стороны, в задачах этого типа исходные растровые изображения имеют достаточно большое количество цветов, обычно превышающее предел, при котором известные алгоритмы дают хорошие результаты, поэтому

дополнительно возникает проблема качественных алгоритмов предварительной обработки растра – уменьшения числа цветов и фильтрации от мелких помех.

Целью данной работы является реализация алгоритмов векторизации для использования в сфере дизайна и полиграфии. Алгоритмы реализованы в виде plugin-модуля для популярной программы Adobe Illustrator, которая предназначена для работы с векторными объектами.

В первой главе производится обзор представленных в литературе методов векторизации, их особенности, достоинства и недостатки. Во второй главе подробно описывается устройство алгоритма векторной скелетизации, использованного при написании программы. В третьей главе дано краткое описание программной реализации в виде подключаемого модуля для Adobe Illustrator.

1. ОБЗОР МЕТОДОВ ВЕКТОРИЗАЦИИ

В настоящее время разработано большое количество разнообразных алгоритмов перевода графических изображений в векторную форму. Одним из классических методов векторизации является распознавание на основе эталонных изображений. Однако этот подход, оправдывающий себя в задачах распознавания печатных и рукописных текстов, мало применим к другим областям, в частности к использованию векторизации в иллюстративной графике, из-за невозможности создания эталонов применительно к данной задаче.

Для данной области больше подходят алгоритмы, в которых делаются попытки выделить на растре некоторые конструкции общего вида. Такими конструкциями на самом нижнем уровне обычно бывают ломаные линии. На последующих этапах возможно дальнейшее уточнение результатов, их последующее распознавание, например, выделение прямых углов, правильных многоугольников, дуг окружностей и др. (т.н. объектная векторизация, см. [1]), либо, применительно к задачам иллюстративной графики, например, аппроксимация найденных ломаных гладкими кривыми.

В алгоритмах, использующих данный подход, задача обычно «решается по следующей схеме: сначала выделяются осевые линии объектов в виде цепочек пикселей растра. Затем выполняется линейная аппроксимация найденных осевых линий, в результате чего формируется модель изображения в виде наборов ломанных, стыкующихся в своих концевых точках. Наконец, выполняется постобработка результата аппроксимации, с целью повышения качества.» [2]

Основные различия алгоритмов обычно заключены в первом этапе выделения осевых линий. В [2] автор выделяет следующие семь групп алгоритмов: (1) основанные на утоньшении линий, (2) основанные на сопоставлении контуров, (3) основанные на графах объектных штрихов, (4) основанные на разбиении изображения регулярной сеткой, (5) основанные на разреженном просмотре растра, (6) основанные на преобразовании Хафа, (7) основанные на аппроксимации объектов растра площадными геометрическими фигурами. Рассмотрим подробнее некоторые из этих подходов:

1. *Методы, основанные на утоньшении линий.* Эти алгоритмы также называют алгоритмами скелетизации или алгоритмами приведения к центральной оси (см. [3]). В результате их работы получается совокупность линий единичной длины, расположенных вдоль центральной оси исходного объекта на растре. Одним из определений скелета является следующее (по Пфалцу и Розенфельду) -

множество всех точек – центров окружностей, помещенных внутрь объекта и имеющих максимально возможные радиусы.

Существует несколько групп алгоритмов, реализующих нахождение скелета.

Алгоритмы, основанные на стирании граничных слоев, решают задачу либо путем итеративного стирания пикселей на границе объекта до получения линии одинарной толщины, либо путем нахождения границ объекта, аппроксимации их ломаными, которые затем рассматриваются как фронт распространения «внутрь объекта» плоской волны. В качестве результата работы в этом случае рассматриваются точки, в которых начинают и заканчивают пересечение различные фронты распространения. Однако алгоритмы данной группы, несмотря на достаточную проработанность, обладают достаточно высокой трудоемкостью и излишней точностью при моделировании осевых линий, если на растровом объекте есть паразитные ответвления, и дают неточности при моделировании пересечений и ответвлений линий на растре (см. рис 1.).

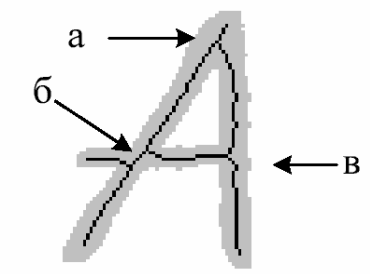


Рис. 1. *а* – паразитные ответвления скелета, *б* и *в* – дефекты при моделировании ветвления и пересечения.

Для кардинального решения проблемы трудоемкости скелетизации различными авторами был предложен ряд неитеративных алгоритмов построения скелета, основанных на определении скелета Пфалца и Розенфельда, которые определяют его как множество центров максимальных кругов, помещенных внутрь объекта. Для выявления данных точек на растре вначале необходимо построить по бинарному растру растр такого же размера, пиксели которого, находящиеся в точках объектных пикселей исходного растра, имеют значения, равные расстоянию от них до ближайшей фоновой точки на исходном растре. Данное расстояние, как правило, вычисляется с использованием некоторой целочисленной метрики, например, расстояния «по Манхэттену». Такое

преобразование получило название преобразования расстояния или дистантного преобразования. Затем требуется просмотреть полученный растр и, точки, в которых наблюдается максимальное значение в локальной окрестности, являются точками, ближайшими к скелетным (которые могут располагаться между пикселями, если толщина линии - четная). Эти алгоритмы имеют трудоемкость строго пропорциональную величине растра. Однако им присущи следующие недостатки: не гарантируется связность скелета, если исходный объект на растре был связным, возможны скелетные линии толщиной в два пикселя, что приводит к усложнению процедур трассировки скелетных линий, данные алгоритмы склонны к ошибкам в позиционировании скелетных точек в местах стыковки или пересечения линий растра.

Для преодоления данных недостатков в литературе были предложены различные алгоритмы, улучшающие качество скелета по Пфалцу-Розенфельду. Но следует отметить, что подобные алгоритмы коррекции увеличивают общую трудоемкость, хотя именно для ее уменьшения в основном и был предложен алгоритм Пфалца-Розенфельда.

2. *Методы, основанные на сопоставлении контуров.* При работе алгоритмы данного класса предполагают, что изображение содержит в основном прямые линии. Производится аппроксимация отрезками границ между объектами. Затем для каждого отрезка делается попытка найти «вторую сторону линии» - другой почти параллельный ему отрезок, ограничивающий область того же цвета, находящийся на расстоянии меньше заданного порога (максимальная ширина линии). После нахождения двух сторон линии, построение осевой линии является тривиальной задачей. Но алгоритмы данного класса с трудом справляются со случаями ответвления линий под малым углом, либо со случаями, когда одному граничному вектору можно сопоставить несколько противоположных векторов, либо когда изображение содержит изогнутые линии. Также недостатком данных алгоритмов является их чрезмерная изощренность при обработке некоторых частных случаев.

3. *Методы, основанные на графах объектных штрихов растра.* Основная идея данной методики – на основе последовательного просмотра растра и анализа последовательностей объектных пикселей в строках и столбцах растра построить

компактное топологического векторное представление растрового изображения. Затем, просматривая это представление, можно достаточно эффективно построить детальную векторную модель растра.

Для этого вводится понятие штриха как последовательности объектных пикселей, расположенных друг за другом в одном столбце или в одной строке, и ограниченной с обоих сторон фоновыми пикселями. Ортогональной координатой штриха называется номер столбца, в котором расположен вертикальный штрих, либо номер строки в случае горизонтального штриха. Путем двукратного просмотра растра находятся все объектные штрихи. После этого производится поиск объектных ребер, как последовательностей смежных штрихов. Найденные объектные ребра приводятся к «скелетному» виду – к линиям одинарной толщины.

Достоинство данной методики – эффективность по времени. К недостаткам данного метода относятся неточная обработка точек ветвления и неточная обработка кривых линий.

4. *Методы, основанные на разбиении изображения регулярной сеткой.* Главным преимуществом алгоритмов данного типа является их высокая временная эффективность. Идея, лежащая в их основе такова: все изображение покрывается регулярной сеткой и анализируются лишь пиксели растра на пересечении с линиями сетки. Содержимое ячейки сетки восстанавливается по конфигурации этих пересечений с использованием предопределенного множества шаблонов (см. рис. 2.).

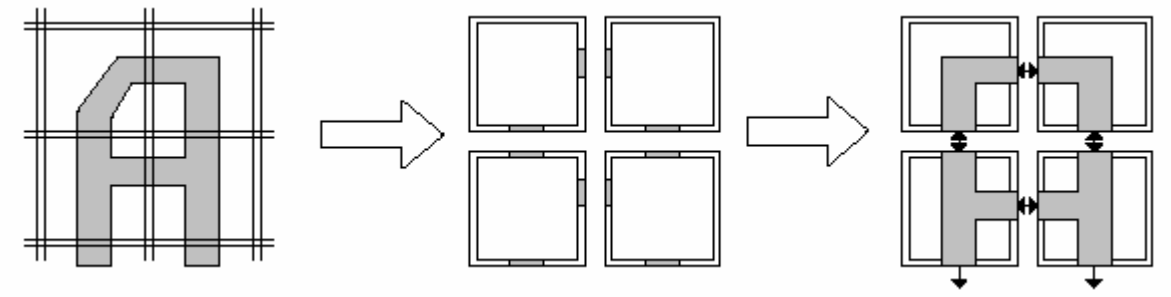


Рис.2. Исходное изображение, разбиение его на квадраты и замена квадратов шаблонами.

При правильном выборе параметров алгоритма большинство ячеек будет определено, оставшиеся же могут быть обработаны либо алгоритмом

скелетизации, либо дальнейшим рекурсивным делением ячейки. К недостаткам данного алгоритма относят сложности при выборе оптимального размера ячейки, которые могут привести к некорректной обработке точек стыковки и ветвления, либо к пропускам линий, в особенности штриховых и пунктирных. Также данные алгоритмы не приспособлены для векторизации изображений с большим числом кривых.

5. *Методы, основанные на разреженном просмотре растра.* Основной целью предложенных алгоритмов является уменьшение трудоемкости. Достигается это за счет отказа от просмотра всего растра, как это производится в алгоритмах скелетизации, производится лишь обработка объектных пикселей, при этом по возможности не всех, а лишь необходимых для построения корректной векторной модели.

Т.н. MIC-алгоритм (алгоритм «максимальных вписанных окружностей», Maximal Inscribing Circles) ориентирован на поиск прямых линий на растре. В идеале прямая линия представляется как некий прямоугольник, имеющий вполне определенные свойства, на которых и основывается метод. При просмотре лишь объектных пикселей алгоритм ищет пиксели, являющиеся центрами вписанных окружностей, таких что границы окружности касаются трех сторон прямоугольника (см. рис. 3). При этом подразумевается, что растр является линейчатым, т.е. не содержит закрашенных площадных объектов, следовательно, каждый прямоугольник представляет собой отдельный сегмент линии.

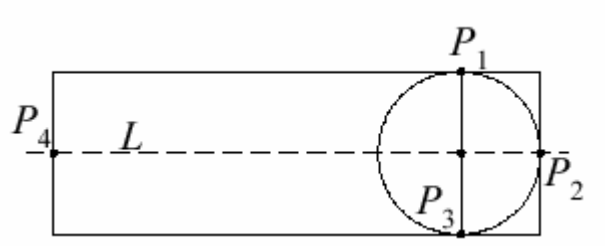


Рис.3.

После нахождения такого центра, найдя две диаметрально противоположные точки касания окружности с фоном (P_1 и P_3), определяем толщину линии как длину отрезка, соединяющего эти точки. Затем производится поиск начала и конца сегмента (P_2 и P_4) как максимально удаленных точек от найденного центра вписанной окружности в направлении оси L найденного сегмента.

Достоинством данного алгоритма является достаточно высокая скорость работы на разреженных линейчатых растрах, в основном состоящих из отрезков прямых линий, но при плотно заполненных растрах, в особенности состоящих из мелких отрезков или плавных кривых, отмечается резкое снижение скорости работы.

Другой алгоритм данного семейства – OZZ-алгоритм восстанавливает среднюю линию объекта следуя траектории пучка света, отражающегося от внутренних границ объекта и перемещающегося вдоль объекта. Средняя линия восстанавливается как ломаная с узлами на серединах траекторий пучка между отражениями.

6. *Методы, основанные на площадном моделировании объектов.* Все вышеизложенные алгоритмы не корректно работают на нелинейчатых растрах. Частичным решением проблемы является сегментация изображения на тонкие линии и площадные объекты с последующей «сшивкой» результатов векторизации, но при этом не всегда возможно совмещение полученных векторных моделей из-за погрешностей, возникающих в процессе оцифровки.

   Для решения данной проблемы были предложены алгоритмы, основанные на следующей идее. Все объекты, изображаемые на исходном растре, аппроксимируются наборами многоугольников, имеющих простую геометрическую структуру, например, квадратами, трапециями и т.п. Затем, в зависимости от характеристик построенных многоугольников, часть из них полагается аппроксимирующей площадные объекты, а оставшаяся часть – линейные. Так как многоугольники имеют простую структуру, то достаточно просто построить векторные модели границ площадных объектов и осевых линий линейных объектов. Из-за того, что для моделирования объектов используется одно множество многоугольников, объединение данных векторных моделей не представляет затруднений.

   Все алгоритмы этого типа имеют трудоемкость большую, чем линейная, относительно площади растра.

«Сравнительное изучение алгоритмов, реализующих вышеописанные методики, позволило сделать следующие выводы. Методы скелетизации являются наиболее универсальным способом получения осевых линий, в отличие от методов сопоставления контуров, разбиения изображения регулярной сеткой, разреженного просмотра растра, а

также методов, основанных на графах объектных штрихов и методов, основанных на преобразовании Хафа, так как позволяют без специальных ухищрений получать осевые линии не только прямых линий, но и произвольных линейчатых растров.» [2]

Но скелетизации присущи такие отрицательные черты, как достаточно большая трудоемкость и ориентированность на бинарные (двуцветные) растры. Теоретически возможно приведение полноцветного растра к набору из одного или более бинарных изображений таким образом, что векторизация полученного набора дает верную общую векторную модель оригинала. Но реализация этого подхода, во-первых, весьма и весьма нетривиальна и, конечно, увеличивает суммарную трудоемкость; во-вторых, приводит к проблемам совмещения векторных моделей отдельных бинарных слоев.

Другим ограничивающим фактором является то, что скелетизация, как и большинство вышеописанных алгоритмов, не позволяет получить правильные результаты при наличии на растре площадных объектов, так как в этом случае векторным представлением площадных объектов будут некие линии, полученные «утоньшением» их до последовательностей пикселей единичной толщины. Естественно для пользователя результаты такой векторизации бесполезны (см. рис 4).

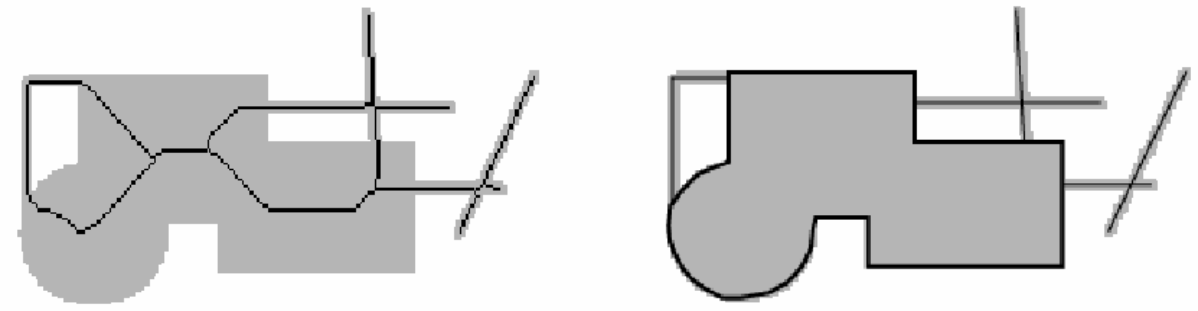


Рис. 4. Слева – результат применения скелетизации к растру, содержащему площадные объекты. Справа – верная с точки зрения пользователя векторная модель растра.

Единственным выходом из данной ситуации является предварительная сегментация растра на части, содержащие объекты одного типа. При этом возможно применение специализированных алгоритмов, например, сегментация текста - графики, линейных - площадных объектов, тонких - толстых линий и пр. (подробнее см. [1]) Для нашей предметной области актуальна лишь сегментация линейных и площадных объектов. Для этого в литературе предлагается подход, основанный на применении операторов математической морфологии, суть которого заключается в следующем: задается пороговое значение толщины, равное n. Далее к изображению применяется преобразование, стирающее до n слоев граничных пикселей. Полученный растр содержит

лишь площадные объекты, его лишь надо восстановить с помощью предусмотренных процедурой методов.

«Недостатком такого алгоритма является то, что ему необходимо выполнить n+1 итерацию завершения работы, таким образом, его трудоемкость пропорциональна $n \cdot S$, где S – площадь исходного изображения» [2]. Кроме этого предполагается, что исходный растр является бинарным, что также ограничивает применение этого алгоритма.

Другой отрицательной стороной сегментации является то, что в результате получаются по сути две векторные модели, которые нуждаются в процессе совмещения, в результате чего может пострадать точность векторизации.

Для преодоления данных недостатков в работах [2] и [4] был предложен алгоритм векторной скелетизации, который позволяет одновременно классифицировать объекты на линейные и площадные и получать единую векторную модель, не требующую дополнительных усилий по «сшивке» объектов. Еще одним его преимуществом является применимость при векторизации многоцветных растров. Данный алгоритм основан на использовании триангуляции в качестве базовой структуры представления содержимого растра. Он и был использован, с небольшими изменениями, для реализации функций векторизации. Подробное его описание приведено в последующих разделах.

2. АРХИТЕКТУРА ПРОГРАММЫ

2.1. ОБЩАЯ СХЕМА РАБОТЫ

Для заданной сферы применения наиболее приемлемым было бы получение векторной модели растра в виде набора кривых различной толщины и площадных объектов. Для построения такой модели был разработан и реализован алгоритм, состоящий из следующих этапов:

- Предварительная обработка растра;
- нахождение граничных линий;
- аппроксимация граничных линий;
- построение триангуляции с ограничениями;
- классификация объектов по триангуляции;
- построение векторных объектов модели.

2.2. ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА РАСТРА

Все алгоритмы векторизации, известные автору, предполагают, что исходный растр либо черно-белый, либо имеет небольшое число цветов. При этом области, окрашенные в различные цвета, считаются различными объектами.

С другой стороны, исходные растры в области иллюстративной графики, дизайна и подготовки бумажных/электронных изданий теоретически могут иметь очень большое число цветов. При этом количество используемых цветов зачастую превышает предел, при котором векторизация имеет смысл. В большинстве случаев применение алгоритмов напрямую к исходным изображениям приведет к созданию очень большого числа векторных объектов, которые не будут соответствовать оригинальному растру в том смысле, что не будут логичными, оправданными с точки зрения пользователя. Например, строка пикселей с плавным переходом цвета будет истолкована как множество отдельных объектов единичного размера, ввиду того, что каждый последующий пиксель будет иметь цвет отличный от предыдущего. Естественно, что такой набор векторных объектов является неверным и бессмысленным для пользователя программы. Ввиду этого необходимо произвести предварительную обработку растра, чтобы собственно алгоритм векторизации имел шанс выделить более логичные с точки зрения пользователя элементы изображения.

2.2.1. УМЕНЬШЕНИЕ КОЛИЧЕСТВА ЦВЕТОВ

Первым этапом предобработки является уменьшение количества цветов. В области машинной графики для этих целей существует специальный класс алгоритмов. Рассмотрим некоторые наиболее известные из них.

Одним из простейших способов является уменьшение количества цветов за счет перехода к палитре, содержащей наиболее часто встречаемые цвета. За счет одного прохода по растру строится гистограмма, по ней выбирается N наиболее часто используемых цветов. После этого еще за один проход все цвета исходного изображения заменяются на наиболее близкие к ним цвета палитры.

Другим распространенным методом является алгоритм «медианного деления»: пусть цвет представлен в виде совокупности значений красной, зеленой и синей компонент, которые принимают значения из интервала [0, 1]. Таким образом, все цвета, представленные на растре, можно разместить в трехмерном пространстве RGB как множество точек внутри единичного куба.

После этого уменьшаем длину каждой из сторон куба до минимальных размеров, чтобы при этом полученная область включала все точки, соответствующие цветам исходного растра. Затем выбираем ту цветовую ось, значения координат точек по которой имеют наибольший разброс. Делим параллелепипед по выбранной оси таким образом, чтобы в каждой части осталось одинаковое число точек. Каждую половину обрабатываем рекурсивно аналогичным образом – редуцируем до объемлющего параллелепипеда путем отрезания «пустых областей», выбираем ось и производим дальнейшее разбиение цветового пространства. Процесс продолжается до тех пор, пока не получим столько параллелепипедов, сколько цветов предполагается получить в конечном растре после постеризации.

На последнем шаге алгоритма находится цвет-замена для цветов, попавших в каждый параллелепипед. Это может быть либо центральная точка полученной области, либо усредненное значение всех входящих в нее точек. Во втором случае трудоемкость алгоритма будет выше, но полученное изображение будет качественнее.

Существуют также множество других алгоритмов постеризации. Но следует отметить, что во всех них задача «классически» формулировалась как получение изображения, наименее отличного от исходного с точки зрения человека. Поэтому качественные алгоритмы постеризации включают в себя дополнительные шаги по

сглаживанию переходов между соседними цветами (т.н. дитеринг, dithering) с помощью таких методов, как наложение шумов и создание специальных либо случайных «узоров» из цветов палитры, которые человеческий глаз с некоторого расстояния воспринимает как цвет, реально не присутствующий в палитре изображения за счет «слитного восприятия» данной области растра.

Но в ходе экспериментов выяснилось, что результат работы классических методов уменьшения количества цветов, даже при исключении шагов алгоритма по сглаживанию полученного в результате первого этапа постеризации изображения, мало подходит для дальнейшей векторизации из-за того, что выбор цвета пикселя конечного растра зависит лишь от цвета пикселя исходного растра, но никак не зависит от его расположения. Поэтому, несмотря на то, что общее число цветов изображения может быть значительно уменьшено (например, при переходе от 65535 цветов к 256 цветам), на результирующем растре очень вероятно появление отличных друг от друга соседних пикселей там, где их векторизация приведет к созданию нелогичных с точки зрения пользователя векторных объектов.

В итоге можно сказать, что классические алгоритмы уменьшения количества цветов на растре изначально создавались для других целей и поэтому их результат мало применим в других областях, и в частности – в задачах векторизации изображений.

Другим методом цветовой предобработки растра, который упоминается в литературе, является применение локальных операторов фильтрации изображения с использованием сглаживающего ядра свертки, т.е. при его применении различие между соседними пикселями уменьшается. Теоретически можно подобрать такое значение элементов ядра, что применение данного фильтра приведет к тому, что будут созданы области одинаковых цветов, которые представляют значимые (логичные с точки зрения пользователя) отдельные элементы изображения. Но при этом этот фильтр производит размывание краев между сильно различающимися цветами, что нежелательно. Этот эффект может быть устранен с помощью локальных фильтров повышения резкости. Но, во-первых, при реализации подобрать значения для матриц фильтров размытия и сглаживания практически невозможно, во-вторых, даже если бы это было сделано, в результате последовательного применения этих двух фильтров появляется возможность смещения границ между областями разных цветов, что, естественно, нежелательно. Поэтому этот подход также был сочтен неподходящим для задач векторизации.

Ввиду всего вышеизложенного для решения задачи цветовой предобработки растра был реализован алгоритм, который с одной стороны не смещает границы между

областями сильно различающихся по значению цветов и, в то же время, стремится к тому, чтобы области сходных цветов были окрашены строго в один цвет и, следовательно, на последующих этапах векторизованы как «цельные» векторные объекты.

Данный алгоритм основан на идее метода «заливки с затравкой» и имеет трудоемкость пропорциональную размерности растра.

1. На очередном шаге алгоритма удаляются все элементы из очереди, выбирается очередной не просмотренный пиксель и заносится в конец очереди. Текущим цветом области считается его цвет. Область считается пустой.
2. В цикле, до тех пор, пока очередь не пуста, производятся следующие действия:
   2.1. Очередной пиксель извлекается из начала очереди. Если он является не просмотренным, то помечается как просмотренный, иначе – не обрабатывается, переход на начало цикла.
   2.2. Если расстояние (в цветовом пространстве) между текущим цветом области и цветом извлеченного пикселя меньше заданного порога, он добавляется в область. Цвет области вычисляется как усредненный цвет всех входящих в него пикселей и приписывается им. Все не просмотренные соседи в 8-ми связной окрестности извлеченного пикселя заносятся в конец очереди.

Расстояние между двумя цветами в пространстве RGB определяется следующим образом:

при следующих обозначениях: (R_1,G_1,B_1) – цвет №1.

(R_2,G_2,B_2) – цвет №2.

расстояние D между цветом №1 и №2 равно:

$$D = 30*(R_2-R_1)^2+59*(G_2-G_1)^2+11*(B_2-B_1)^2.$$

Множители 30, 59, 11 отражают различную чувствительность человеческого глаза к красной, зеленой и синей компонентам цвета соответственно.

Единственный параметр вышеприведенного алгоритма – порог расстояния между двумя цветами, при превышении которого считается, что данные два пикселя принадлежат разным областям. Этот параметр задается пользователем программы, его изменение позволяет регулировать «подробность» получаемой векторной модели – чем

меньше порог, тем больше останется областей разного цвета, тем больше будет создано разных векторных объектов. На рисунке 5 показаны результаты работы алгоритма.

Рис.5. Слева – результат постеризации без применения дитеринга. Справа – результат работы предложенного алгоритма – при его применении создается меньше соседних областей различного цвета.

Особенностью данного алгоритма является то, что нельзя сказать, сколько цветов было оставлено на изображении. Более того, вполне вероятно создание множества весьма сходных, но не равных по значению цветов. Т.е. строго говоря, данный алгоритм имеет целью не уменьшение числа используемых цветов вообще, а уменьшения числа различных цветов у соседних пикселей.

2.2.2. ФИЛЬТРАЦИЯ ШУМОВ

Полученное на предыдущем этапе изображение может содержать одиночные (имеющие цвет отличный от цветов всех соседей) пиксели – если на исходном растре они были окрашены в цвет, который находится на большем расстоянии от цветов всех соседних пикселей, нежели заданный пользователем порог. В результате они будут считаться отдельными областями единичного размера. Такие пиксели считаются шумом и удаляются с векторизуемого изображения за один просмотр.

2.3. ВЫДЕЛЕНИЕ ГРАНИЧНЫХ ЛИНИЙ

Входом для данного этапа является растр G размерностью (M x N). Выходом – множество ломаных, разделяющих области различного цвета.

Для нахождения граничных линий необходимо построить вспомогательный растр – растр границ. Для его построения вначале расширим растр G, добавив две строки: до первой и после последней и аналогично – со столбцами. Значения добавленных строк и

столбцов – некий «нулевой» цвет, отличный от всех цветов растра G. Теперь сформируем растр границ B(u[i, j], b[i, j]) размерностью (M+1 x N+1). По смыслу элементы растра располагаются «по углам» пикселей исходного растра G.

«Элемент b[i, j] растра B задает границы между четырьмя соседними пикселями различных цветов на растре G, а элемент u[i, j] – признак не удаляемой узловой точки. В элементе b[i, j] границы кодируются четырьмя битами следующим образом: если есть линия от центра вправо - то код «1000»; если вверх - то код: «0100»; если влево - то код: «0010»; если вниз – то код: «0001». При нескольких линиях общий код образуется наложением кодов по операции «или». Если цвета всех 4-х пикселей одинаковы, то код равен 0. Элемент u[i, j] == 1, если узловая точка в центре между четырьмя соседними пикселями есть, u[i, j] == 0, если узловой точки нет» [2], (см. рис. 6). Узловая точка – точка, в которой соседствуют три или больше пикселя разных цветов, причем пиксели одного цвета, если они есть, не расположены по диагонали относительно друг друга. Очевидно, что растр B может быть построен путем однократного просмотра входного изображения.

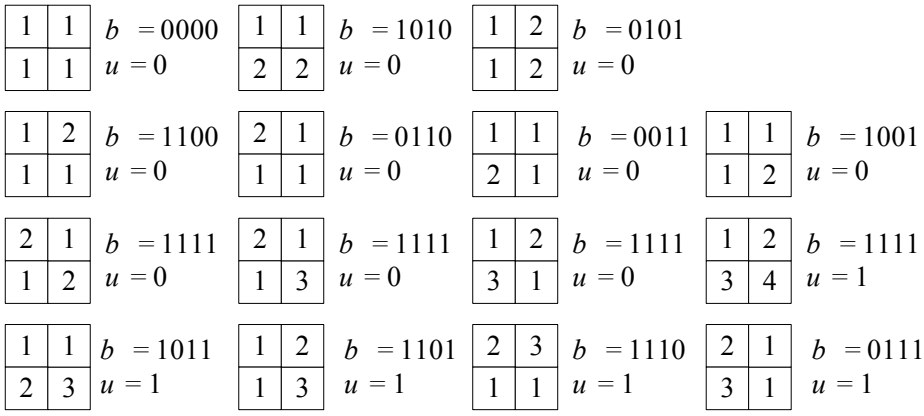


Рис. 6.

В [4] дается следующее определение граничной линии: «граничной линией на растре B назовем такую ломаную линию, образованную точками растра, что соседние узлы этой ломаной являются 4-соседями на растре B, при движении вдоль линии от начального узла до конечного слева и справа от всех отрезков ломаной находятся пиксели растра G двух разных цветов (слева одного, справа – другого цвета), причем ломаная является максимальной по включению».

Граничная линия может быть двух типов – замкнутой или открытой. Исходя из этого, алгоритм начинает последовательный просмотр растра границ B, до тех пор пока не будет найден ненулевой элемент b[i, j]. Этот элемент считается входным, начиная с него

начинается отслеживание граничной линии до тех пор, пока алгоритм не замкнет линию либо пока не придет в узловую точку. Если отслеживание граничной линии закончилось в узловой точке, то необходимо произвести аналогичную трассировку в обратную сторону из входной точки b[i, j]. В процессе трассировки очередная точка добавляется в ломаную, если она является начальной или конечной, либо не находится на горизонтальной или вертикальной прямой, проходящей через предыдущую и следующую точки (т.е. в ней граничная линия меняет направление движения).

Во время трассировки граничной линии при проходе через точку B[i, j] обнуляются флаги направлений входа и выхода b[i, j]. В исходной точке обнуляется лишь бит выхода, в конечных – биты входа.

В большинстве случаев направление дальнейшего движения по растру границ B в процессе отслеживания линии не возникает проблем неоднозначности. Единственный случай, который нуждается в особой обработке – при «шахматном расположении» пикселей (когда b[i, j] == “1111”, u[i, j] == “0”). В подобных случаях неоднозначность может быть разрешена, если считать, что объектные пиксели всегда находятся справа от граничной линии и, следовательно, в подобных ситуациях необходимо совершать «поворот налево», чтобы не нарушить 8-ми связность объекта.

Так как просмотр растра B производится слева направо, сверху вниз, то очевидно, что входной точкой может быть точка вида: b[i, j] == “1000” или “0001” или “1001”, u[i, j] == “0” или “1”, причем варианты, когда b[i, j] == “1000” или “0001” возможны лишь при u[i, j] == “1”.

Назовем границей объекта на растре совокупность граничных линий, состоящую либо из одной замкнутой линии, либо из множества последовательно расположенных (последняя точка предыдущей совпадает с первой точкой следующей) линий, образующих замкнутый контур, ограничивающий область одного цвета на исходном растре.

Процедура нахождения границы объекта состоит в следующем: из исходной точки делается попытка движения «вправо» (“1000”). Исключением является случай, когда текущим столбцом является N+1 – очевидно, что в этом случае необходимо двигаться «вниз». Если данное направление является возможным, то производится отслеживание граничной линии. Если найденная линия является замкнутой, то объект выделен. Иначе необходимо продолжить отслеживание границы объекта из последней точки найденной граничной линии таким образом, чтобы цвет справа по направлению движения остался прежним. Продолжаем до тех пор, пока не получим замкнутый контур.

Если такое направление движения невозможно, следовательно такая граничная линия уже была отслежена при нахождении границы другого объекта (ввиду направления обхода, все объекты исходного растра, расположенные выше текущей строки и левее текущего столбца уже полностью выделены и обработаны) – в этом случае необходимо обратиться к хранилищу уже найденных граничных линий и запросить искомую по известной точке начала и цветам слева и справа. Очевидно, что эти параметры однозначно идентифицируют любую граничную линию. Каждая граничная линия может принадлежать максимум двум объектам. Дальнейшее отслеживание продолжается с последней точки этой линии до тех пор, пока контур не замкнется.

Такое упорядочение обхода позволяет получить набор более «логичных» граничных линий, по сравнению с произвольным выбором направления движения и цвета отслеживаемого объекта.

2.4. АППРОКСИМАЦИЯ ГРАНИЧНЫХ ЛИНИЙ

Полученные на предыдущем этапе граничные линии содержат очень большое число точек и чрезмерно подробны. Поэтому необходимо аппроксимировать результаты предыдущего шага. При этом отклонения от исходной ломаной должны быть минимизированы, причем желательно, чтобы значения отклонений по обе стороны от аппроксимирующей линии были максимально близки по значениям.

Сначала на граничной линии выделяются характерные узловые точки (см. рис.7). Это точки, расположенные:

1. в отрезках, являющихся локальными экстремумами, при условии, что ни один из концов отрезка не является угловой узловой точкой (см. след. пункт);
2. в углах – местах стыковки двух отрезков прямых, если длины обоих строго больше 2 пикселей, либо строго равны 2 пикселям;
3. на отрезках длины больше 2 пикселей, за 0.5 пикселя от места стыковки с другим отрезком, если длина второго отрезка строго равна 2 пикселям.

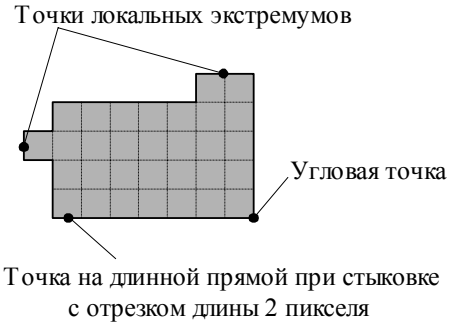


Рис.7 Характерные точки замкнутой граничной линии

Для работы алгоритма первичной аппроксимации необходимо, чтобы каждая граничная линия содержала как минимум две узловые точки. Для незамкнутых линий это условие выполняется изначально, т.к. точки начала и конца линий естественно являются узловыми. Если замкнутая граничная линия не содержит характерных узловых точек, то в качестве узловых выбираются, например, две точки глобальных максимумов по обеим координатам.

Затем исходная линия преобразуется следующим образом:

Шаг 1: В середину каждого отрезка исходной ломаной вставляется дополнительная точка, если это не было сделано на предыдущем этапе – при вставке узловых точек локальных экстремумов.

Шаг 2: В ходе последовательного просмотра очередная точка ломаной сохраняется, если она является узловой или добавленной на шаге 1, иначе – помечается как подлежащая удалению.

Шаг 3: Все помеченные на шаге 2 точки удаляются из линии.

После этого производится последовательный просмотр участков граничной линии, заключенных между узловыми точками. Для каждых двух узловых точек A и B, расположенных последовательно (не считая не узловые точки), находится точка C, имеющая максимальное значение отклонения от прямой AB и, если оно больше или равно 0.5 пикселя, то данная точка считается узловой, а отрезки AC и CB обрабатываются аналогичным образом, иначе осуществляется переход к следующей паре узловых точек.

Теорема: При аппроксимации согласно вышеописанному алгоритму максимальное отклонение полученной аппроксимированной ломаной от исходной составляет величину менее 1 пикселя.

Доказательство:

Введем следующие обозначения (см. рис. 8):

- X – точка исходной ломаной;
- a, b – инцидентные точке X отрезки исходной ломаной;
- N, M – середины отрезков a и b соответственно;
- d – сегмент аппроксимирующей линии;
- K, T – точки пересечения d с отрезками a и b соответственно.

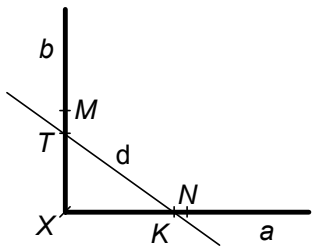


Рис. 8.

Рассмотрим следующие случаи:

1. ||a|| > 2 и ||b|| > 2 либо ||a|| = 2 и ||b|| = 2. Согласно правилам выделения характерных узловых точек на предварительном этапе аппроксимации (пункт №2) точка X будет помечена как узловая и войдет в состав аппроксимирующей ломаной. Следовательно, отклонение в данном случае равно 0.
2. Длина одного из отрезков строго равна 2, длина другого – больше 2 пикселей. Пусть для определенности ||a|| = 2, ||b|| > 2. Тогда, согласно пункту №3, на предварительном этапе аппроксимации на отрезке b, на расстоянии 0.5 пикселя от точки X будет добавлена узловая точка, через которую пройдет аппроксимирующая ломаная. Очевидно, что расстояние от точки X до этой ломаной не может быть больше 0.5 пикселя.
3. Длина одного из отрезков равна 1 пикселю. Пусть для определенности ||a|| = 1. Тогда согласно правилам построения аппроксимирующей ломаной, K находится на расстоянии < 0.5 от точки N. Следовательно, расстояние от X до ближайшей из точек пересечения отрезка d с отрезками a и b менее 1 пикселя. Следовательно, евклидово расстояние от точки X до отрезка d составляет величину также менее 1 пикселя.

Т.е. во всех случаях отклонение от любой вершины ломаной, а следовательно, и от любой точки на ней, составляет величину меньше 1 пикселя.

Теорема доказана.

Для хранения данных необходима дискретность представления координат, равная 0.5 пикселя, что может быть легко обеспечено в рамках целочисленной арифметики – достаточно хранить координаты удвоенными.

2.5. ПОСТРОЕНИЕ ТРИАНГУЛЯЦИИ С ОГРАНИЧЕНИЯМИ

Следующим этапом работы является построение триангуляции Делоне с ограничениями. В качестве ребер ограничений выступают отрезки граничных линий, полученные на предыдущем шаге.

Сначала по множеству точек, строится триангуляция Делоне. За линейное время строится произвольная триангуляция, после чего происходит перестроение тех треугольников, для которых не выполняется условие Делоне.

Затем происходит вставка ребер ограничений, при этом используется следующий подход:

1. Находится область D, состоящая из треугольников, пересекаемых вставляемым ребром ограничения. Все треугольники, составляющие область D, удаляются из триангуляции;
2. После этого производится пересоздание треугольников внутри данной области таким образом, чтобы ребро ограничения гарантировано было добавлено:
    2.1. Производится проход отдельно по левой границе области D из вершины, в которой начинается вставляемое ребро ограничения до вершины, в которой оно заканчивается. В процессе прохода по границе для каждых трех последовательно расположенных точек проверяется возможно ли по ним построить треугольник. Если да, то он создается;
    2.2. Шаг 2.1 повторяется до тех пор, пока не будет создан треугольник, содержащий вставляемое ребро ограничения;
    2.3. – 2.4. – Аналогично обрабатывается правая граница области D.

Очевидно, что на каждом шаге 2.1. или 2.3. создается как минимум один новый треугольник, т.к. граница области D – замкнута. Таким образом, алгоритм закончит работу, когда заполнит область D треугольниками произвольным образом, но при этом гарантируется вставка ребра ограничения.

В результате внутри таких областей могут нарушаться условия Делоне, возможно появление сильно вытянутых треугольников, например, зачастую такие треугольники входят в состав линейных объектов и этим обусловлена их неправильная форма.

Следующий этап – раскраска триангуляции в соответствии с цветами слева и справа от найденных граничных линий.

Шаг 1: всем треугольникам, содержащим ребра ограничений, присваиваются цвета в соответствии с цветом справа и цветом слева от граничной линии, к которой принадлежит ребро ограничения.

После этого на *шаге 2* необходимо просмотреть триангуляцию в поисках треугольников, у которых не был определен цвет. Это т.н. «внутренние треугольники», которые не содержат ребер ограничений. Цвет каждого из них определяется цветом любого «внешнего» треугольника, который либо является соседним для данного «внутреннего» треугольника, либо которого можно достичь переходя от соседа к соседу по «внутренним» треугольникам от данного.

2.6. РАЗМЕТКА ТРИАНГУЛЯЦИИ

Для дальнейшей работы необходимо произвести разделение объектов на линейные и площадные. Применительно к данной реализации эту задачу можно сформулировать как классификацию треугольников на треугольники, входящие в состав линейных объектов и на треугольники, входящие в состав площадных объектов. Далее первые треугольники будем условно называть «линейными», а вторые – «площадными».

2.6.1. ВЫДЕЛЕНИЕ ЛИНЕЙНЫХ ОБЪЕКТОВ

Очевидным критерием классификации объекта является толщина объекта. Если она превышает некоторый заданный порог, то объект считаем площадным, иначе – линейным. Но точное измерение толщины объекта по триангуляции является достаточно изощренной и трудоемкой операцией, поэтому используются различные эвристические методы.

В работе [2] предложен следующий подход: рассматриваются пары треугольников, имеющих общее невидимое ребро. Для всех возможных конфигураций таких пар треугольников определяются методы измерения толщины объекта. На основе данных измерений и с помощью задания двух порогов (минимальной толщины площадного объекта и максимальной ширины линейного) и производится разметка треугольников.

Данный подход достаточно сложен в реализации и требует вычисления различных параметров треугольников для разных пар (в одном случае – средняя длина медиан, для другой пары треугольников – средняя высота из двух точек, в третьем – длина ребра).

В качестве более простого в реализации алгоритма был предложен алгоритм первичной разметки треугольников на основании соотношения площади области к суммарной длине ребер ограничений, входящих в область. А более точное измерение толщины объекта производить для тех треугольников, которые на первом этапе помечены

как линейные.

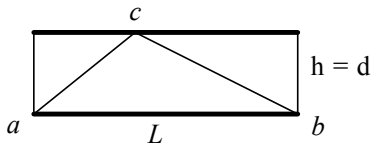


Рис.9.

Рассмотрим идеальный случай: отрезок прямой линии длины L (см. рис.9). Треугольник abc содержит одно ребро ограничения (ab) длины L. Высота треугольника равна h = d – ширине линии.

$$S_{abc} = \frac{1}{2} \cdot h \cdot L$$
$$S_{линии} = h \cdot L$$
$$S_{abc} = \frac{1}{2} \cdot S_{линии}$$

Такие соотношения также выполняются с некоторыми погрешностями для случаев вида, изображенных на рис.10, с условием, что площадь внутреннего треугольника A тоже будет учтена.

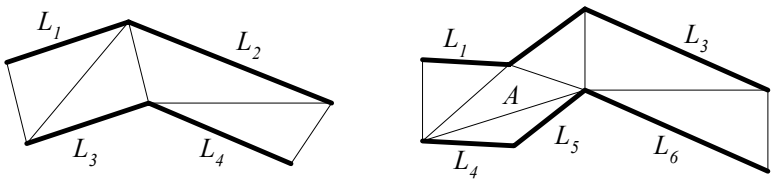


Рис. 10.

Таким образом, можно сформулировать следующий критерий: если площадь треугольника меньше половины произведения суммарной длины ребер ограничений на максимально возможную ширину линии, то треугольник считается линейным.

Однако вычислительные эксперименты показывают, что применение этого критерия к отдельным треугольникам дает неудовлетворительные результаты, алгоритм дает неприемлемо большое количество ошибочных площадных треугольников внутри линейных объектов, из-за мелких погрешностей или особых частных случаев триангуляции линейного объекта – когда создаются треугольники, являющиеся тупоугольными и сильно вытянутыми.

Необходимо выделять некие связные области одного цвета на триангуляции и применять критерий ко всем треугольникам области одновременно (суммируя их площади и длины ребер ограничений, входящих в состав выделенных треугольников). Но при выборе области нужно учитывать, что ситуации, когда линия одного цвета переходит в полигональный объект того же цвета также должны быть корректно обработаны, и зоны перехода должны выделяться достаточно качественно. Именно из-за этого неприменим «кумулятивный подход» - когда в область добавляются треугольники пока критерий выполняется и, когда больше нельзя продолжать процесс, все треугольники области считаются линейными. Это связано с тем, что линия на растре может быть меньшей толщины, чем максимальная, в результате при достаточно большой длине наблюдается эффект «накопления свободной площади» - когда за счет экономии площади на линейных треугольниках в линию включаются площадные треугольники, несмотря на то, что критерий для них по отдельности не выполняется.

Поэтому предлагается формировать области для проверки как множество треугольников, смежных с некой заданной опорной вершиной. Причем, если область удовлетворяет критерию линейной, то всем ее треугольникам присваивается тип «линейный», но если она не удовлетворяет критерию, это вовсе не значит, что все треугольники – площадные. Плюсом данного подхода является то, что отдельный треугольник может быть проверен несколько раз, включен в 3 разные области и это увеличивает вероятность того, что он будет классифицирован правильно. Лишь после того, как будут проверены все варианты включения треугольников в линейные объекты, т.е. в качестве опорных точек для подобластей триангуляции будут испробованы все существующие точки, оставшиеся неклассифицированными треугольники помечаются как входящие в состав площадных объектов.

Если область треугольников помечена как часть линейного объекта, то по ее значениям площади и суммарной длины ребер ограничений можно наоборот вычислить предполагаемую толщину линейного объекта в данной окрестности и использовать эту информацию при построении объектов векторной модели растра.

В процессе вычислительных экспериментов было выяснено, что этот подход дает гораздо более качественные результаты, как по сравнению с методом применения критерия к отдельным треугольникам (ошибки нераспознавания линейных треугольников), так и по сравнению с «кумулятивным» подходом (ошибки включения в линию площадных треугольников).

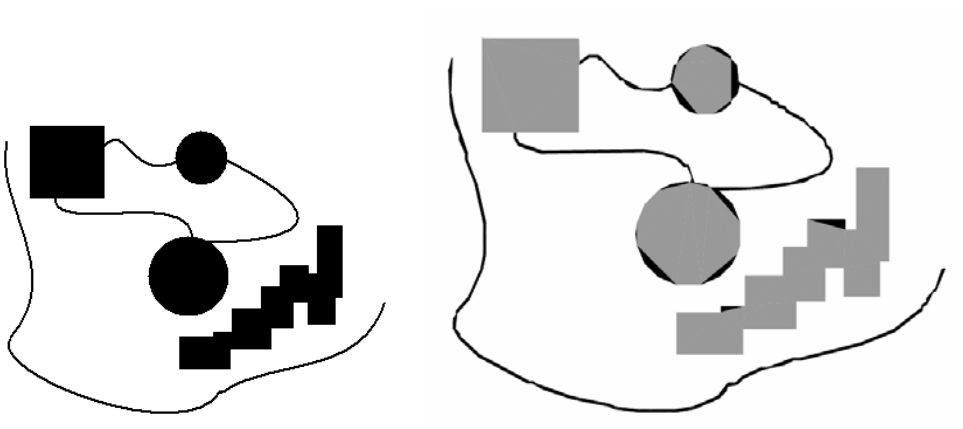


Рис. 11. Результат работы алгоритма классификации объектов. Исходный растр и проанализированная триангуляция. Черным цветом изображены треугольники, классифицированные как линейные, серым – как площадные.

Недостатком данного алгоритма является то, что он не учитывает того, что длина границы между площадным и линейным объектом не может быть больше определенного порога. В результате появляются ошибки, явно видные на рис. 11.

Для исправления ошибок данного типа используются дополнительные проверки – сразу после проведения вышеописанного этапа классификации и на этапе построения линейных объектов по триангуляции.

Первая проверка – с помощью одного линейного просмотра триангуляции все линейные треугольники, у которых суммарная длина открытых ребер больше максимальной ширины линии d, считаются ошибочно классифицированными как линейные и помечаются как площадные. Открытым ребром треугольника назовем такое ребро, не являющееся ребром ограничения, что треугольник, расположенный по другую сторону от него, имеет тип (линейный или площадной), не равный типу данного треугольника.

Этот этап позволяет исправить ошибки классификации, при которых одиночные треугольники на границе площадных объектов принимаются алгоритмом за линии (см. рис.12).

Трудоемкость данного этапа – линейная относительно числа точек в триангуляции. Так как число треугольников линейно зависит от числа точек. Следовательно, получаем линейную зависимость трудоемкости от числа треугольников в триангуляции.

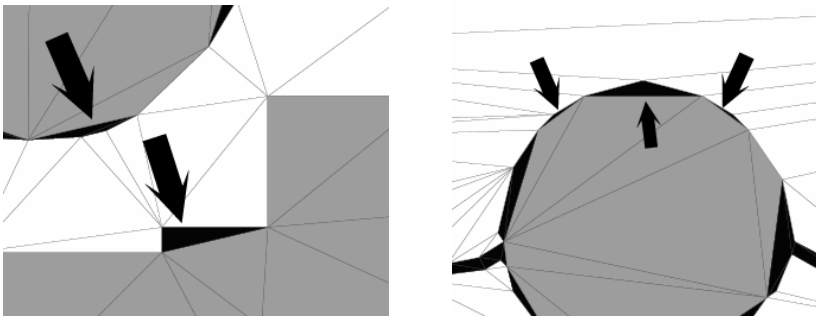


Рис. 12.

2.6.2. ВЫДЕЛЕНИЕ ПЛОЩАДНЫХ ОБЪЕКТОВ

После проведения выделения линейных объектов все оставшиеся непомеченными треугольники считаются принадлежащими к площадным объектам. Но это разделение не является окончательным, т.к. уже на этапе создания объектов по размеченной триангуляции могут быть обнаружены ошибки, допущенные на предыдущем этапе. Поэтому создание площадных векторных объектов необходимо производить лишь после создания линейных векторных объектов (см. 2.7).

Вообще говоря, было бы желательно, произвести дополнительную проверку всех непомеченных треугольников перед тем, как приписать им тип площадных, т.к. ошибки, когда линейные треугольники помечаются как площадные, хотя и редко, но встречаются. Но на данном этапе не было найдено приемлемых критериев такой проверки.

2.7. ПОСТРОЕНИЕ ВЕКТОРНЫХ ОБЪЕКТОВ

Результатом работы алгоритмов предыдущих этапов является полностью размеченная триангуляция, содержащая достаточную информацию для построения векторной модели содержимого растра.

При создании векторного представления линейных объектов производится поиск первого треугольника, имеющего тип линейного. С него начинается отслеживание смежных линейных треугольников одного цвета с одновременным построением скелетной линии.

В зависимости от количества ребер ограничений, либо открытых ребер, входящих в состав линейного треугольника, треугольники можно условно разделить на 3 типа: *«внутренние»* (все ребра являются невидимыми), *«боковые»* (в состав треугольников входит одно ребро ограничения и два невидимых ребра), *«оконечные»* (состоят из одного невидимого ребра и двух ребер ограничений).

Алгоритм в процессе работы строит участки скелетных линий, каждый из которых является либо замкнутым, либо начинается и заканчивается в *«оконечном»* или *«внутреннем»* треугольнике, либо в *«боковом»* треугольнике, одно из невидимых ребер которого открытое, т.е. по другую сторону от него – площадной треугольник. Скелетные линии проходят через середины невидимых ребер *«боковых»* треугольников, заканчиваются в точке пересечения медиан, если последний треугольник линии – *«внутренний»*, или в точке пересечения ребер ограничений в случае *«оконечного»* треугольника, либо на середине открытого ребра, если последний треугольник данного участка скелетной линии – *«боковой»* - см. рис. 13 – случаи соответственно обозначенные как 1, 2, и 3.

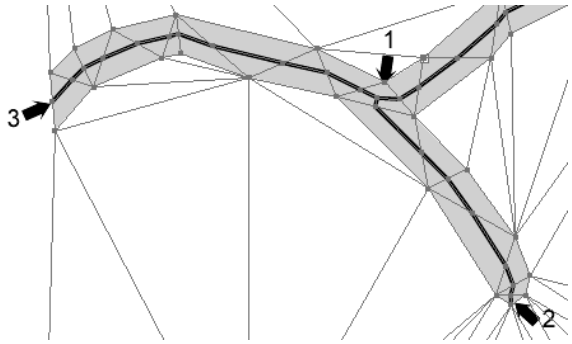


Рис. 13.

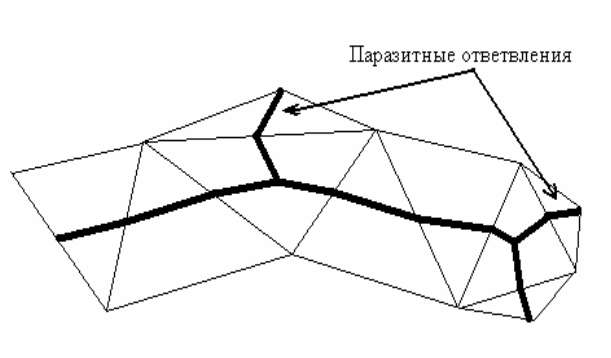


Рис. 14.

После получения набора ломаных необходимо отфильтровать паразитные ответвления (рис. 14) – удалить линии малой длины (с длинами порядка максимальной толщины линии), выходящие из узлов с валентностью три.

Затем проводится второй этап фильтрации ошибок классификации треугольников. Для всех ломаных, один конец которых расположен в *«оконечном»* треугольнике, а другой – в *«боковом»* с одним открытым ребром, вычисляется длина (для повышения скорости используется метрика «по Манхэттену») и суммарная длина открытых ребер – т.е. длина границы между линией и полигоном. Если оказывается, что эти два значения близки по величине, то треугольники, через которые проходит линия, считаются ошибочно классифицированными как линейные, они помещаются как площадные, линия удаляется (см. рис. 15).

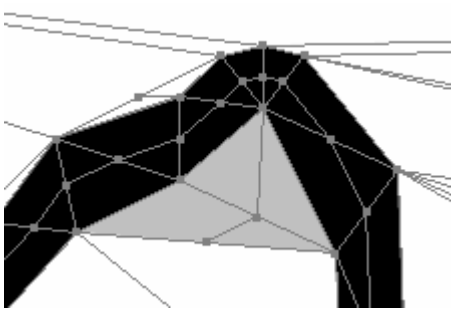


Рис. 15. Серым цветом выделены треугольники, ошибочно классифицированные как линейные. Длина построенной по ним линии соизмерима с суммарной длиной открытых ребер треугольников – поэтому линия будет удалена, а треугольники помечены как площадные.

После этого необходимо «склеить» смежные участки скелетных линий, при условии, что они проходят через треугольники одного цвета и ширина их различается на величину меньше некоторого заданного порога. Иначе данная точка считается точкой стыковки двух линий разной толщины.

Для создания векторных представлений площадных объектов необходимо построить максимальные по включению связные области, состоящие только из треугольников, не являющихся линейными. Векторным представлением данного объекта является внешняя граница области. Ее нахождение по триангуляции является очевидной процедурой.

Трудоемкость данного этапа – линейная относительно числа треугольников, так как при построении объектов каждый треугольник должен быть просмотрен не более чем 1 раз. Все процедуры фильтрации – линейны относительно числа созданных векторных объектов. Векторных объектов невозможно создать больше, чем построено треугольников.

3. РУКОВОДСТВО ПРОГРАММИСТА

3.1. ПРИНЦИПЫ ВЗАИМОДЕЙСТВИЯ ПРОГРАММЫ С ADOBE ILLUSTRATOR

Продукт Adobe Illustrator является одним из наиболее распространенных и известных векторных редакторов в своей области. По сути эта программа является одним из стандартов de facto в области иллюстративной векторной графики.

Данный программный продукт предоставляет возможности расширения функциональности с помощью подключаемых модулей, т.н. plugins, формат которых является открытым и, таким образом, любой сторонний разработчик может добавлять новые возможности к основной программе. По сути, сам Adobe Illustrator во многих своих частях и подсистемах реализован как набор подключаемых модулей.

Подключаемый модуль представляет собой динамически загружаемую библиотеку с именем вида *.aip, которая помещается в специальную поддиректорию программы. При старте Adobe Illustrator загружает все найденные модули и производит их инициализацию.

Взаимодействие основной программы с plugin производится через посылку ему сообщений путем вызова объявленных форматом модуля функций с заданным интерфейсом. Какие конкретные сообщения будут посланы модулю зависит от его типа. Архитектурой Adobe Illustrator предусмотрены следующие типы plugin'ов:

- ***Action*** – модули этого типа используются для автоматического выполнения команд меню Illustrator по заданной программе. Модуль также может зарегистрировать себя для того, чтобы дать возможность записывать макрокоманды.
- ***File format*** – модули этого типа используются для того, чтобы расширить число входных и/или выходных типов файлов, которые может распознать основная программа.
- ***Filter*** – модули этого типа регистрируются в меню программы Menu→Filters→… и предназначены для создания и/или манипулирования объектами оригинал-макета, загруженного в Illustrator. Подразумевается, что фильтр предоставляет некоторый интерфейс пользователю для того, чтобы тот мог установить параметры перед выполнением действий.

- ***Menu command*** – модули этого типа используются для добавления новых элементов в структуру меню основной программы. Обычно такие plugin используются для добавления подпунктов в меню Menu→Window→Hide/Show window.
- ***Notifier, Timer*** – модули этих типов извещаются о наступлении неких событий. Timer plugin получает уведомления через заданный интервал времени, а Notifier plugin может запросить уведомление о различных изменениях в основной программе, например, о том, что был выбран другой элемент в текущем документе. Регистрация на получение различного вида уведомлений происходит при загрузке и инициализации plugin.
- ***Plugin group*** – модули этого типа поддерживают одну или несколько plugin group. Plugin group – это объект специального вида, который содержит как редактируемые части, так и те, которые отображаются, но не подлежат редактированию со стороны пользователя. Такие подключаемые модули отвечают за генерацию изображения group plugin при каждом его изменении. Они используются для создания таких объектов как “Live Blends” и “Brushes”.
- ***Tool*** – соответствующая модулю кнопка добавляется на палитру инструментов Adobe Illustrator, а сам модуль получает возможность при активации пользователем отслеживать перемещение указателя мыши, определять выбранный объект оригинал-макета и как-либо воздействовать на него. Например, Tool plugin мог бы случайным образом искривлять объекты или создавать новые объекты на основе указанных пользователем.
- ***Suite*** – модули этого типа предоставляют новые наборы функций – т.н. suite. О них подробнее будет изложено ниже.

Для целей реализации векторизатора наиболее подходящим является тип “Filter”, поэтому работа с остальными разновидностями подключаемых модулей далее рассматриваться не будет.

Итак, при загрузке Adobe Illustrator вызывает функцию, являющуюся точкой входа в библиотеку:

```
SPAPI SPErr PluginMain(char *caller, char *selector, void *data);
```

При этом все три аргумента, передаваемые в функцию, вместе составляют сообщение. Первые два параметра описывают что модуль должен выполнить. Переменная caller

содержит имя отправителя сообщения и категорию действия. A selector определяет конкретное действие внутри данной категории, которое необходимо предпринять.

Любой подключаемый модуль может получать следующие пять сообщений: reload, unload, startup, shutdown и about. Также модуль может получать дополнительные сообщения в зависимости от его типа. К примеру, plugin который публикует новые suites будет получать acquire и release сообщения при запросах клиентов.

Таблица 1. Основные сообщения, посылаемые подключаемому модулю от Adobe Illustrator.

Caller	**Selector**	**Ожидаемые в ответ действия plugin**
kSPAccessCaller (“SP Access”)	kSPAccessReloadSelector (“Reload”)	Восстановить информацию о состоянии программы
	kSPAccessUnloadSelector (“Unload”)	Сохранить информацию о состоянии программы
kSPInterfaceCaller (“SP Interface”)	kSPInterfaceStartupSelector (“Startup”)	Инициализировать данные о состоянии программы, зарегистрироваться в Illustrator
	kSPInterfaceShutdownSelector (“Shutdown”)	Освободить память, выделенную программе, уведомить Illustrator о выгрузке, сохранить внутренние настройки во внешних ресурсах
	kSPInterfaceAboutSelector (“About”)	Показать диалог с информацией о подключаемом модуле.

Третий аргумент, передаваемый в main-функцию библиотеки, содержит указатель на структуру данных, которая содержит информацию, соответствующую присланному сообщению. Например, когда происходит получение сообщения о нажатии кнопки мыши, данная структура будет содержать координаты курсора, и т.д. В то время как содержимое структуры данных сообщения меняется, первые три поля являются общими для всех типов сообщений и объявлены как:

```
typedef struct {
SPPluginRef self;
void *globals;
SPBasicSuite *basic;
} SPMessageData;
```

Поле `self` содержит ссылку на вызываемый подключаемый модуль. Оно используется при регистрации suites предоставляемых plugin, адаптеров и других данных в Adobe Illustrator.

Поле `globals` содержит указатель, предназначенный для использования самим подключаемым модулем. Обычно он указывает на область памяти, выделенную программе и используемую для сохранения данных между вызовами. Эта область, как правило, создается при инициализации модуля и сохраняется системой Illustrator неизменной в промежутке времени между событиями kSPAccessCaller/Reload и kSPAccessCaller/Unload.

Поле `basic` является указателем на объект AIBasicSuite, через который происходит доступ к использованию остальных suites. Понятие suites рассматривается позднее.

Вышеописанные сообщения могут быть получены любыми подключаемыми модулями независимо от их типа. Кроме них существуют сообщения, которые посылаются только Filter-plugin. Это сообщения kCallerAIFilter/kSelectorAIGetFilterParameters и kCallerAIFilter/kSelectorAIFilterGo. Первое сообщение посылается, когда система Adobe Illustrator предоставляет подключаемому модулю возможность вывести графический диалог пользователю для запроса у него параметров, установок режимов работы и т.д. Если в ответ на это сообщение не возвращается ошибки, то система вслед посылает сообщение kCallerAIFilter/kSelectorAIFilterGo, сигнализируя о том, что ожидается запуск основного алгоритма модуля и получение от него каких-либо результатов. Сообщения запроса параметров и начала работы разделены для поддержки функции “Apply Last Filter” – когда пользователь может выбрать данный пункт меню для запуска фильтра с параметрами, указанными при последнем использовании. В этом случае plugin получит сразу лишь второе сообщение из вышеописанной пары, без запроса на вывод GUI и изменение параметров.

Преимуществом модульной архитектуры Adobe Illustrator для программиста является то, что многие функции и механизмы уже реализованы, например печать документов, повтор/отмена действий пользователя, поддержка истории команд, функции работы с цветами, слоями, многие математические преобразования, декодирование

разных форматов входных файлов, сохранение в файл и т.д. В результате разработчик может сосредоточиться на реализации собственно своего алгоритма.

Доступ к ресурсам, функциям системы Adobe Illustrator осуществляется через т.н. suites. Suite это объект, предоставляющий доступ к указателям на различные функции AI API. Каждый отдельный suite содержит интерфейс к отдельной подсистеме основной программы, например, AIRealMathSuite позволяет выполнять различные математические преобразования с числами с плавающей точкой, AIArtSuite предоставляет наиболее общий интерфейс доступа к объектам текущего документа, для создания, изменения, либо удаления объектов и т.д.

Для каждого suite ведется подсчет ссылок для оптимизации управления памятью. Перед использованием функций какого-либо suite необходимо запросить его у системы с помощью `AcquireSuite` и после завершения работы с ним – уведомить об этом систему с помощью вызова `ReleaseSuite`. В свою очередь эти функции (`AcquireSuite и ReleaseSuite`) входят в состав AIBasicSuite, ссылку на который получает любой подключаемый модуль при инициализации и при получении любого сообщения (см. выше).

Кроме этого Adobe Illustrator предоставляет средства для построения платформо-независимых пользовательских интерфейсов с помощью т.н. Adobe Dialog Manager (ADM). ADM реализован как набор siute'ов, функции которых используются при работе с элементами пользовательского интерфейса – для запроса значений элементов управления, их изменения, для обработки различных реакций и создания взаимосвязей между элементами управления на формах и т.д. Для каждой платформы создаются лишь различные файлы ресурсов – код программы же остается неизменным в большинстве случаев.

3.2. ОБЩАЯ СХЕМА РЕАЛИЗАЦИИ ПРОГРАММЫ

Для реализации системы был выбран язык C++ ввиду его широких функциональных возможностей, высокой эффективности результирующего кода и наличия общепризнанных стандартов (International Standard for Information Systems – Programming Language C++, ANSI C), что существенно облегчает перенос программы на другие платформы (это актуально, так как в области дизайна, полиграфии и смежных областей широко используются платформы, отличные от x86, например, Apple Macintosh). В качестве среды разработки использовалась Microsoft Visual Studio 6.

На рис. 16 изображена схема взаимодействия основных классов программы. Как видно из диаграммы, класс CVectorizer является основным связующим звеном и реализует «фасад» подсистемы векторизации для остальных компонентов, обеспечивая максимально простой интерфейс - вызов единственной функции после заполнения структуры параметров алгоритма. После этого он либо сам реализует подсистемы и механизмы собственно оцифровки растра, либо инициирует их исполнение другими классами.

Экземпляр класса CVectorizer существует лишь во время работы алгоритма оцифровки растра и во время работы пользователя с графическим интерфейсом системы. Этот экземпляр инстанцируется при вызове функции FilterGetParams(…) (который происходит при получении plugin сообщения kCallerAIFilter/kSelectorAI-GetFilterParameters), или при вызове GoFilter(…) (вызывается в ответ на сообщение kCallerAIFilter/kSelectorAIFilterGo) и уничтожается при выходе из этих функций.

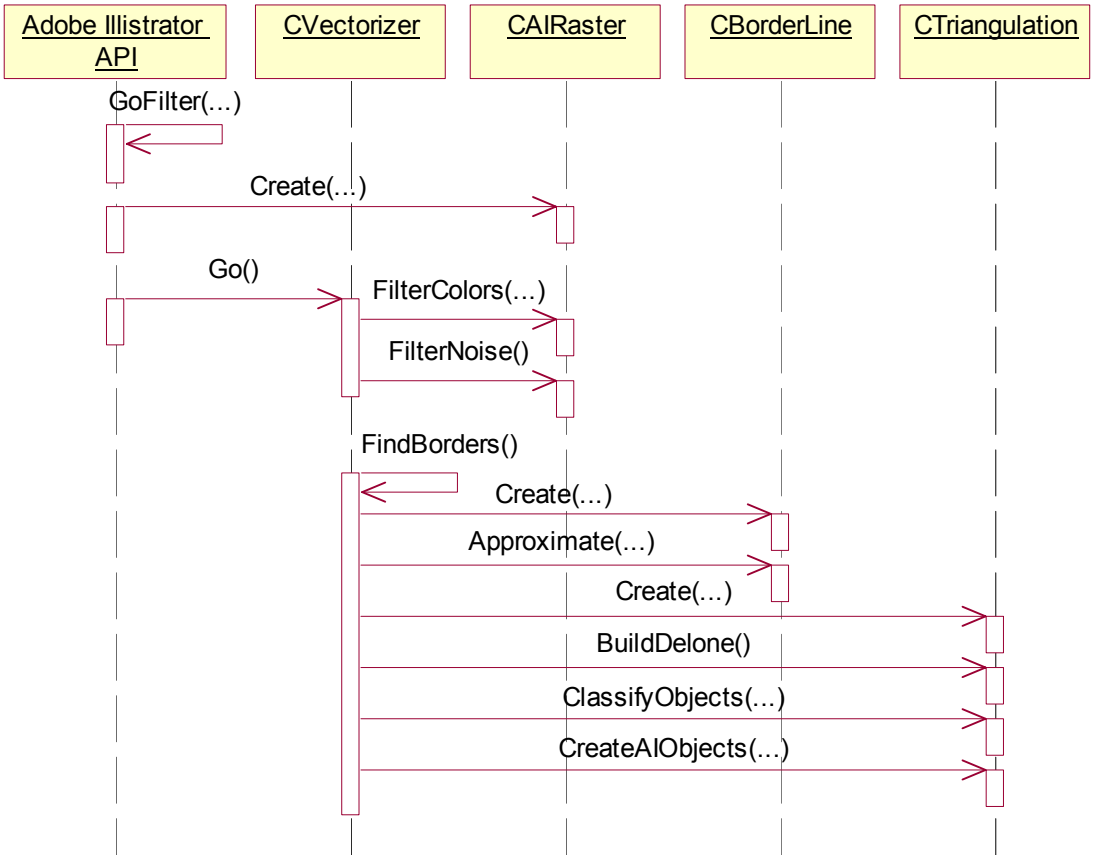


Рис. 16. Диаграмма последовательностей запуска алгоритмов векторизации.

FilterGetParams(…) находит выбранный объект среди всех объектов открытого в данный момент документа Adobe Illustrator, проверяет его корректность (что он является растровым изображением поддерживаемого подключаемым модулем формата) и

инстанцирует объект класса CVectorizer, параметризуя его указателем на объект, представляющий выбранный растр. Ссылка на созданный экземпляр сохраняется в глобальной переменной. Затем создается немодальный диалог для настройки пользователем параметров алгоритма. При нажатии на кнопку “Generate preview” создается дополнительный слой в иерархии объектов текущего документа, открытого в Adobe Illustrator, в котором и строится векторное представление входного изображения – выполняется запуск функции CVectorizer::Go(). При нажатии кнопок “Ok” или “Cancel” диалоговое окно закрывается, только во втором случае происходит удаление созданного слоя со всеми его объектами.

В случае вызова функции GoFilter(…) происходит лишь нахождение выбранного объекта в текущем документе Adobe Illustrator, его проверка на корректность и осуществляется запуск алгоритма. Такой вариант возможен, когда пользователь запускает фильтр не из меню Menu→Filter→Vectorizer, а через Filter→Apply Last Filter, при этом используются настройки, сохраненные во время предыдущего старта.

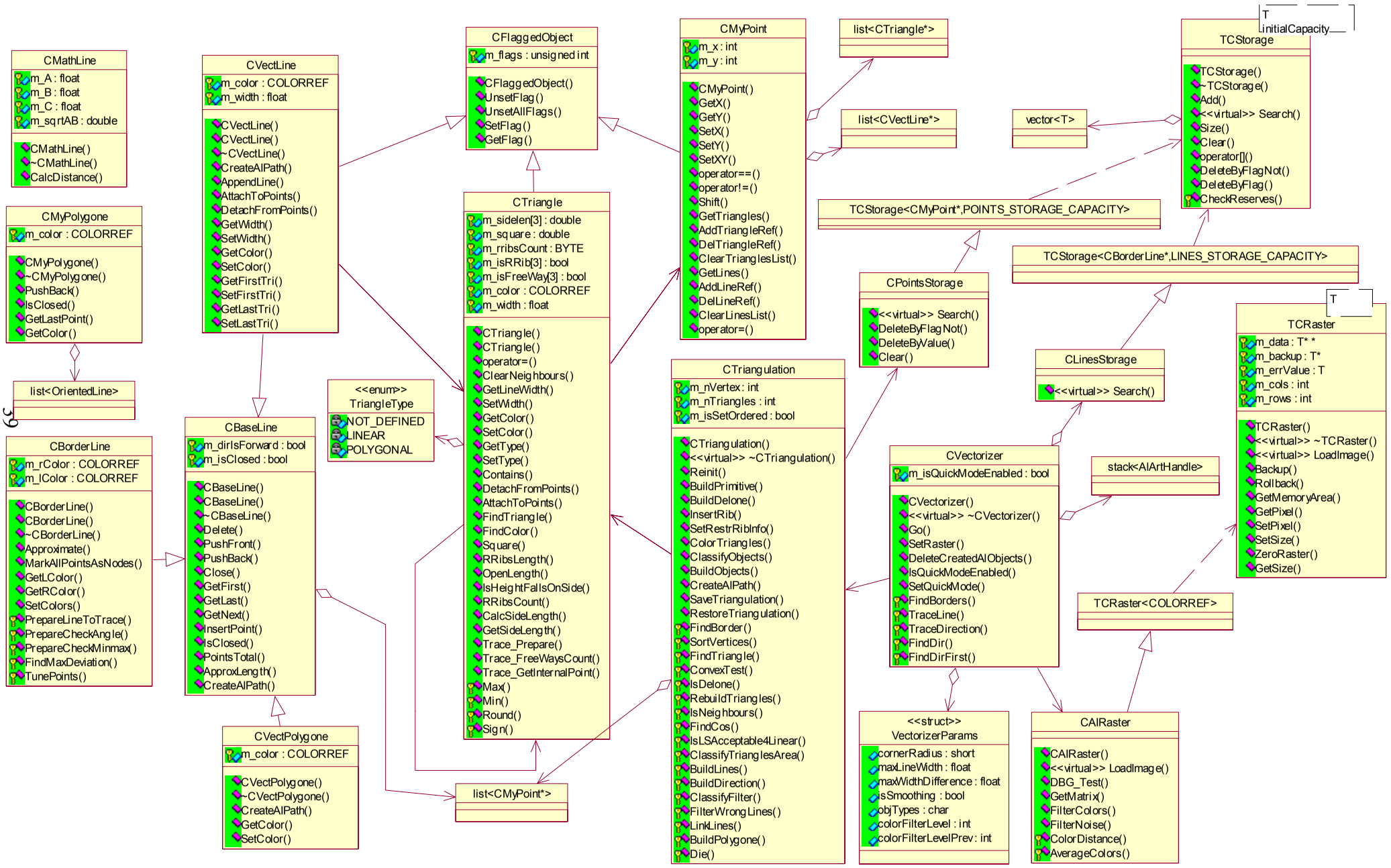

CMathLine
m_A : float
m_B : float
m_C : float
m_sqrtAB : double
CMathLine()
~CMathLine()
CalcDistance()
CVectLine
m_color : COLORREF
m_width : float
CVectLine()
CVectLine()
~CVectLine()
CreateAlPath()
AppendLine()
AttachToPoints()
DetachFromPoints()
GetWidth()
SetWidth()
GetColor()
SetColor()
GetFirstTri()
SetFirstTri()
GetLastTri()
SetLastTri()
CFlaggedObject
m_flags : unsigned int
CFlaggedObject()
UnsetFlag()
UnsetAllFlags()
SetFlag()
GetFlag()
CMyPoint
m_x : int
m_y : int
CMyPoint()
GetX()
GetY()
SetX()
SetY()
SetXY()
operator==()
operator!=()
Shift()
GetTriangles()
AddTriangleRef()
DelTriangleRef()
ClearTrianglesList()
GetLines()
AddLineRef()
DelLineRef()
ClearLinesList()
operator=()
list<CTriangle*>
list<CVectLine*>
T
initialCapacity
TCStorage
TCStorage()
~TCStorage()
Add()
<<virtual>> Search()
Size()
Clear()
operator[]()
DeleteByFlagNot()
DeleteByFlag()
CheckReserves()
vector<T>
TCStorage<CMyPoint*,POINTS_STORAGE_CAPACITY>
TCStorage<CBorderLine*,LINES_STORAGE_CAPACITY>
CMyPolygone
m_color : COLORREF
CMyPolygone()
~CMyPolygone()
PushBack()
IsClosed()
GetLastPoint()
GetColor()
list<OrientedLine>
CTriangle
m_sidelen[3] : double
m_square : double
m_rribsCount : BYTE
m_isRRib[3] : bool
m_isFreeWay[3] : bool
m_color : COLORREF
m_width : float
CTriangle()
CTriangle()
operator=()
ClearNeighbours()
GetLineWidth()
SetWidth()
GetColor()
SetColor()
GetType()
SetType()
Contains()
DetachFromPoints()
AttachToPoints()
FindTriangle()
FindColor()
Square()
RRibsLength()
OpenLength()
IsHeightFallsOnSide()
RRibsCount()
CalcSideLength()
GetSideLength()
Trace_Prepare()
Trace_FreeWaysCount()
Trace_GetInternalPoint()
Max()
Min()
Round()
Sign()
<<enum>>
TriangleType
NOT_DEFINED
LINEAR
POLYGONAL
CPointsStorage
<<virtual>> Search()
DeleteByFlagNot()
DeleteByValue()
Clear()
CLinesStorage
<<virtual>> Search()
T
TCRaster
m_data : T* *
m_backup : T*
m_errValue : T
m_cols : int
m_rows : int
TCRaster()
<<virtual>> ~TCRaster()
<<virtual>> LoadImage()
Backup()
Rollback()
GetMemoryArea()
GetPixel()
SetPixel()
SetSize()
ZeroRaster()
GetSize()
CBorderLine
m_rColor : COLORREF
m_lColor : COLORREF
CBorderLine()
CBorderLine()
~CBorderLine()
Approximate()
MarkAllPointsAsNodes()
GetLColor()
GetRColor()
SetColors()
PrepareLineToTrace()
PrepareCheckAngle()
PrepareCheckMinmax()
FindMaxDeviation()
TunePoints()
CBaseLine
m_dirIsForward : bool
m_isClosed : bool
CBaseLine()
CBaseLine()
~CBaseLine()
Delete()
PushFront()
PushBack()
Close()
GetFirst()
GetLast()
GetNext()
InsertPoint()
IsClosed()
PointsTotal()
ApproxLength()
CreateAlPath()
CTriangulation
m_nVertex: int
m_nTriangles : int
m_isSetOrdered : bool
CTriangulation()
<<virtual>> ~CTriangulation()
Reinit()
BuildPrimitive()
BuildDelone()
InsertRib()
SetRestrRibInfo()
ColorTriangles()
ClassifyObjects()
BuildObjects()
CreateAlPath()
SaveTriangulation()
RestoreTriangulation()
FindBorder()
SortVertices()
FindTriangle()
ConvexTest()
IsDelone()
RebuildTriangles()
IsNeighbours()
FindCos()
IsLSAcceptable4Linear()
ClassifyTrianglesArea()
BuildLines()
BuildDirection()
ClassifyFilter()
FilterWrongLines()
LinkLines()
BuildPolygone()
Die()
CVectorizer
m_isQuickModeEnabled : bool
CVectorizer()
<<virtual>> ~CVectorizer()
Go()
SetRaster()
DeleteCreatedAlObjects()
IsQuickModeEnabled()
SetQuickMode()
FindBorders()
TraceLine()
TraceDirection()
FindDir()
FindDirFirst()
stack<AlArtHandle>
TCRaster<COLORREF>
<<struct>>
VectorizerParams
cornerRadius : short
maxLineWidth : float
maxWidthDifference : float
isSmoothing : bool
objTypes : char
colorFilterLevel : int
colorFilterLevelPrev: int
CAlRaster
CAlRaster()
<<virtual>> LoadImage()
DBG_Test()
GetMatrix()
FilterColors()
FilterNoise()
ColorDistance()
AverageColors()
CVectPolygone
m_color : COLORREF
CVectPolygone()
~CVectPolygone()
CreateAlPath()
GetColor()
SetColor()
list<CMyPoint*>

Рис. 17.

3.3. ОПИСАНИЕ КЛАССОВ И ФУНКЦИЙ ПРОГРАММЫ

На рисунке 17 изображена общая диаграмма классов приложения. Дадим краткое описание основных классов подсистемы векторизации.

- Класс CVectorizer – с одной стороны служит фасадом всей подсистемы векторизации, с другой – инкапсулирует реализацию одной из ее функциональных частей – нахождение граничных линий. Наиболее значимые атрибуты этого класса:
  - m_pRaster – содержит указатель на объект класса CAIRaster, который инкапсулирует информацию об обрабатываемом изображении;
  - m_linesStore – содержит указатели на построенные граничные линии растра в процессе работы функции FindBorders();
  - m_pTriangulation – указатель на создаваемый в процессе работы объект класса CTriangulation, которому делегируются все функции по построению и анализу триангуляции;
  - s_params – статически объявленная структура данных, содержащая все необходимые настройки алгоритма векторизации – порог цветовой фильтрации, максимально допустимая ширина линии, режим работы, параметры сглаживания полученных ломаных линий и т.д.
- Параметризованный класс TCRaster – представляет собой матрицу (растр) элементов заданного типа. Обеспечивает общие функции доступа и манипулирования элементами и атрибутами, поддерживает механизм сохранения текущего состояния растра/отката к сохраненному варианту – используется, например, для того, чтобы пользователь мог увидеть результаты работы программы при разных значениях параметров – для этого нужно сохранить исходный растр, чтобы перед началом работы с измененными значениями настроек можно было бы вернуться к начальному состоянию;
- Класс CAIRaster – является потомком шаблона TCRaster, специализированного типом COLORREF, для представления растровых изображений в формате RGB. Данный класс реализует механизмы загрузки данных через Adobe Illustrator API, алгоритмы цветовой предобработки исходного изображения – уменьшения количества цветов (метод FilterColors(…)) и фильтрация мелких помех (метод FilterNoise()). Специфичный для класса атрибут – m_rasterMatrix, содержащий в себе матрицу преобразований (сдвиг/масштабирование/поворот), примененных к растру.

Т.к. векторная модель строится во внутренних координатах растра без учета внешних преобразований, поэтому для того, чтобы представить пользователю результат работы в том же масштабе и положении как и исходный растр, необходимо хранить эту информацию.

- Класс CTriangulation – инкапсулирует алгоритмы построения триангуляции Делоне с ограничениями, а также все используемые в процессе векторизации алгоритмы, основанные на ее применении (разметка объектов, построение скелетных линий, площадных объектов). Основные методы и атрибуты класса:
  - m_nTriangles, m_nVertex – количество треугольников и вершин в триангуляции;
  - m_T, m_V – массивы треугольников и вершин триангуляции;
  - m_saveT – указатель на область сохранения массива треугольников. Используется для ускорения работы – если пользователь в процессе подбора нужных настроек изменил лишь те параметры, которые касаются только этапов анализа триангуляции, то нет необходимости заново производить трудоемкие вычисления предыдущих этапов, достаточно просто восстановить триангуляцию в ее «исходном» виде и произвести разметку с новыми значениями параметров.
  - BuildDelone() – процедура постройки триангуляции. Сначала производится построение произвольной триангуляции, затем – перестроение треугольников которые не удовлетворяют критерию Делоне;
  - BuildLines(…) – процедура построения векторных линейных объектов по размеченной триангуляции. Для каждого найденного треугольника, помеченного как линейный, запускается трассировка для построения скелета, определения средней толщины линии. В результате создаются экземпляры класса CVectLine;
  - BuildPolygone(…) – процедура построения векторной модели площадного объекта как внешней границы составляющих его смежных площадных треугольников. Для представления построенных объектов используются экземпляры класса CVectPolygone;
  - ClassifyObjects(…) – метод, реализующий разделение треугольников на линейные и площадные согласно алгоритму изложенному в п.2.6. В процессе работы использует функцию ClassifyTrianglesArea(…), которая применяет

критерий разделения к множеству треугольников, переданному в качестве параметра;

- Функции ClassifyFilter(…) и FilterWrongLines(…) – реализуют описанные в пп. 2.6., 2.7. методы коррекции ошибок разметки и создания векторных объектов;
- LinkLines(…) – производит сшивку между собой сегментов линейных объектов, найденных на первом этапе построения векторного представления содержимого растра;
- InsertRib(…) – метод, реализующий вставку ребра ограничения в триангуляцию;
- SetRestrRibInfo(...) – метод, предназначенный для приписывания цветов треугольникам согласно информации о цветах слева и справа от граничных линий. ColorTriangles(…) – метод, реализующий распространение информации о цветах на «внутренние» треугольники с помощью «заливки с затравкой».

- Класс CBaseLine и его потомки – CBorderLine, CVectLine и CVectPolygone – представляют объекты-ломаные. Первый подкласс дополнительно реализует алгоритмы аппроксимации найденных граничных линий по алгоритму, изложенному в п.2.4., второй и третий – используются для построения и представления готовой векторной модели в виде набора ломаных линий и полигонов, а также для отображения их в структуру документа Adobe Illustrator – при вызове метода CreateAIPath(…) по переданному дескриптору объекта документа основной программы создается соответствующий объект в иерархии Adobe Illustrator с необходимым цветом, толщиной, масштабом, смещением и углом поворота. Внутреннее представление во всех трех классах – двусвязный список точек ломаной линии;
- Параметризованный класс TCStorage – представляет собой хранилище объектов произвольного типа. Параметрами класса являются тип хранимых объектов и начальная емкость. При полном заполнении происходит перераспределение памяти с увеличением емкости хранилища вдвое. Внутреннее представление – шаблон vector из библиотеки STL;
- Класс CTriangle – класс, представляющий треугольник триангуляции. Основные атрибуты класса:
  - m_T[3], m_V[3] – массивы вершин и соседних треугольников;

- m_type – перечисление, задающее тип треугольника (линейный, площадной, неопределенный);
- m_square – переменная, хранящая площадь треугольника (сохраняется для ускорения работы);
- m_color – цвет, приписанный треугольнику.

- Класс CMyPoint – описывающий точку ломаной, либо вершину триангуляции. Обеспечивает точность представления координат 0.5 пикселя в рамках целочисленных переменных.

ЗАКЛЮЧЕНИЕ

В ходе проведенной работы были изучены различные алгоритмы векторизации растровых изображений с целью получить представление о сильных и слабых сторонах различных подходов и выбрать наиболее подходящий алгоритм для заданной сферы применения – области дизайна и иллюстративной графики. В процессе изучения были предложены и апробированы новые способы решения задач сегментации растра на площадные и линейные объекты, первичной аппроксимации граничных линий в контексте использования алгоритма векторной скелетизации, предложенного в работах [2] и [4].

В результате была написана программа, реализующая функции автоматической векторизации полноцветных растровых изображений с возможностью выделения линий и полигонов. В данный момент программа находится на этапе тестирования и после исправления некоторых технических ошибок и проведения оптимизации примененных алгоритмов, может быть представлена в виде коммерческого продукта на рынке подключаемых модулей для программного обеспечения Adobe Inc.

Автор работы ознакомился с задачами векторизации и предложенными в литературе методами ее решения, приобрел опыт написания подключаемых модулей для систем Adobe Inc.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Обработка и отображение информации в растровых графических системах.- Минск: ИТК АН БССР.- 1989 г.-180 с.
2. Новиков Ю.Л. Эффективные алгоритмы векторизации растровых изображений и их реализация в геоинформационной системе. Дисс.на соиск.уч.степ.к.т.н. - Томск: Томский гос.ун-т.- 2002 г.
3. Розенфельд А. Распознавание и обработка изображений с помощью вычислительных машин /Пер.с англ.- М., Мир,1972 г. - 230 с..
4. Костюк Ю.Л., Новиков Ю.Л. Графовые модели цветных растровых изображений высокого разрешения //Вестник ТГУ, 2002 г., №275, с.153 -160.

ПРИЛОЖЕНИЕ А. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ПРОГРАММЫ ВЕКТОРИЗАЦИИ

На рис. 18. изображен пользовательский интерфейс программы. С помощью данного диалога можно настроить следующие параметры работы алгоритма:

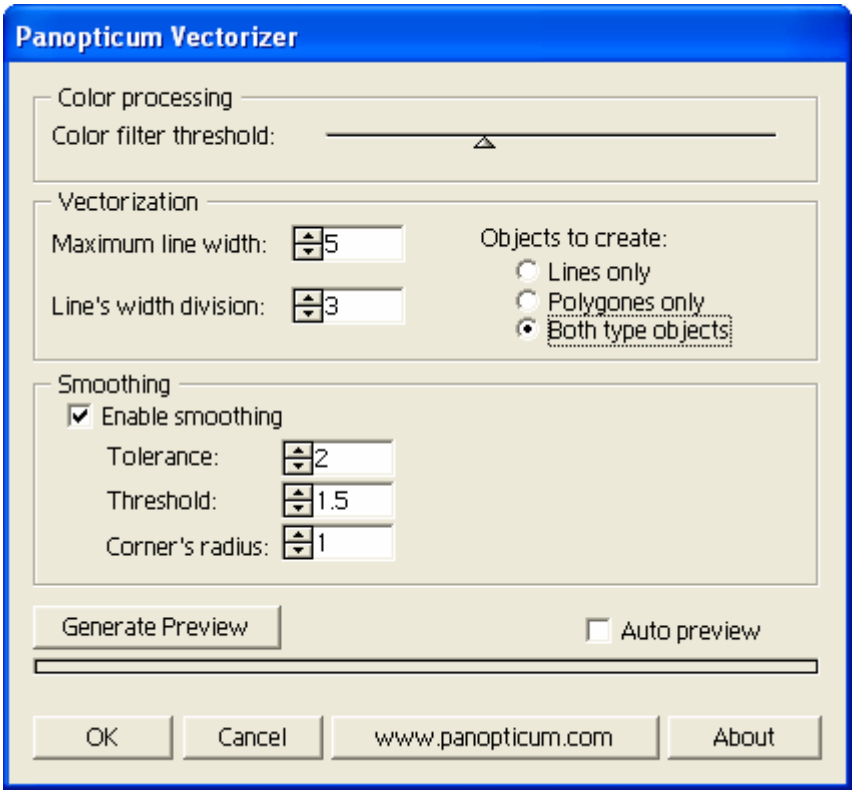


Рис.18. Диалог настроек программы.

Секция: Color processing (Цветовая предобработка).

Параметр: Color filter threshold (Порог фильтрации цвета) – определяет количество различных цветов, которые будут учитываться алгоритмом. Чем больше значение параметра, тем меньше различных цветов будет содержать результат работы программы. Соответственно, уменьшение порога ведет к построению большего числа векторных объектов. Следует помнить, что программа трактует каждую отдельную область некоторого цвета как отдельный векторный объект (как полигон или линию), поэтому в результате векторизации многоцветных растров с низким значением порога цветовой фильтрации возможно создание очень большого числа векторных объектов, что может привести к существенному замедлению работы Adobe Illustrator или к ошибке нехватки памяти в системе.

Секция: Vectorization (Векторизация).

Параметр: Objects to create (Тип создаваемых объектов) – задает режим работы векторизатора. При выборе варианта “Lines only” («Только линии») любая область одного цвета будет усредняться неким образом до одной или нескольких линий; в случае выбора варианта “Polygons only” («Только полигоны») программа будет создавать лишь площадные объекты на основе областей различного цвета; при выборе третьего режима (“Both type objects”, «Объекты обоих видов») программа будет автоматически анализировать вид отдельной области и, в зависимости от ее толщины, создавать объект того или иного типа.

Параметр: Maximum line width (Максимальная ширина линии) – определяет пороговую величину толщины (в пикселях) объекта, который будет трактоваться при обработке изображения как линия, если включен режим работы “Objects to create – Both type objects”. При остальных режимах этот параметр игнорируется.

Параметр: Line’s width division (Разделение линий по толщине) – контролирует процесс сшивки смежных линий. Если при векторизации будут созданы две линии, соединяющиеся в одной точке, то они будут объединены в одну, если разница толщин у них меньше заданной величины. При объединении толщина линии усредняется. Значение параметра используется, если включен режим работы “Objects to create – Both type objects”. При остальных режимах этот параметр игнорируется.

Секция: Smoothing (Сглаживание).

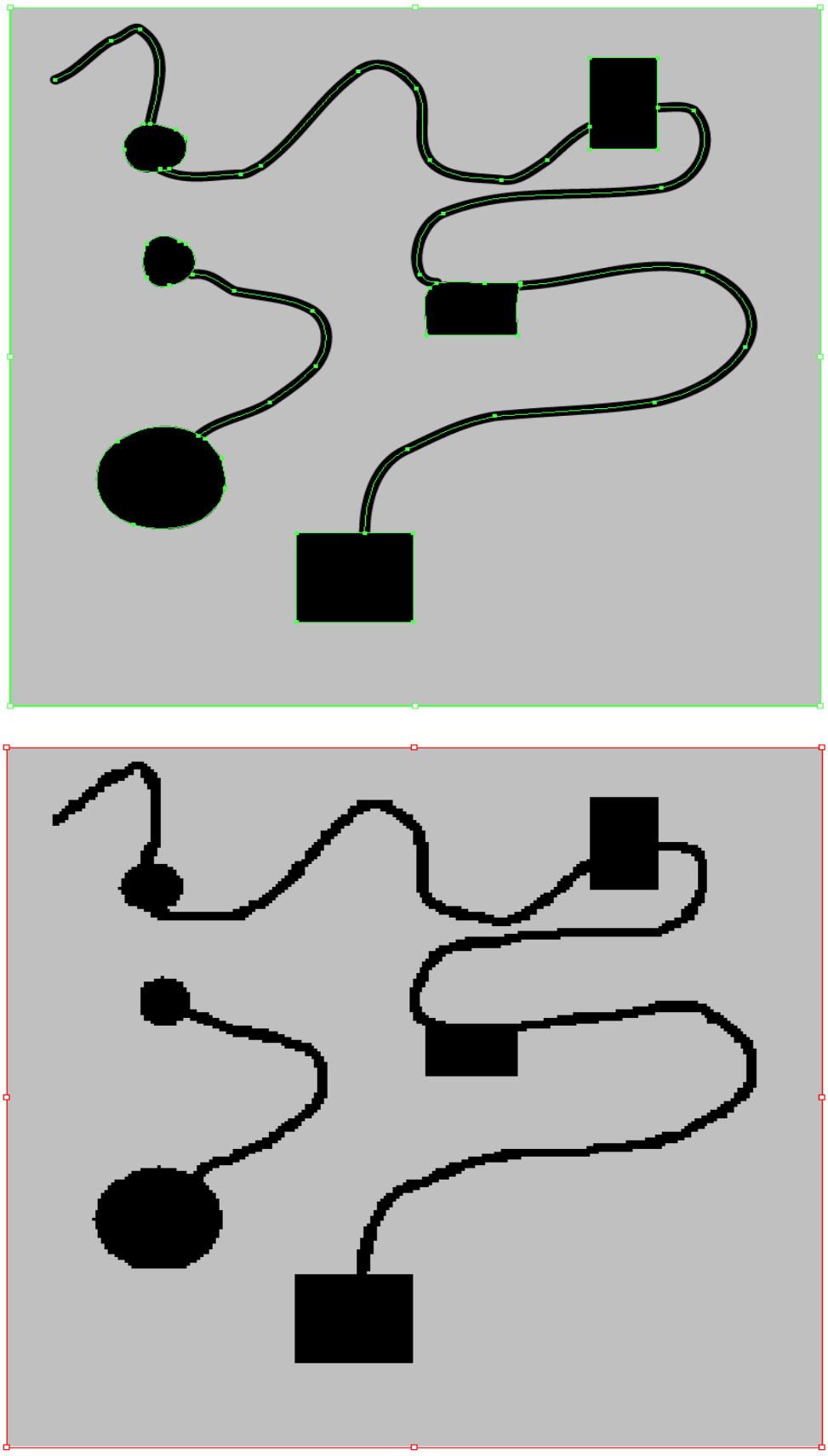
Параметр: Enable smoothing (Включить сглаживание) – регулирует вид результата векторизации – будет ли он представлен в виде ломаных и многоугольников, либо программа будет создавать объекты на основе гладких кривых.

Параметры: Tolerance, Threshold, Corner’s radius – эта тройка определяет насколько близко будет проходить кривая от точек исходной ломаной и какой вид будет иметь.

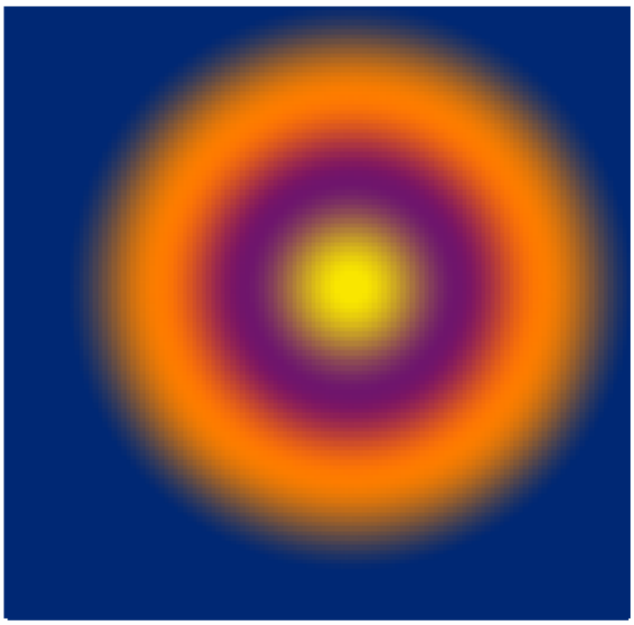
Чтобы вызвать диалог необходимо выбрать соответствующий пункт меню: Menu→Filter→Vectorizer. Данный пункт становится активным только если в данный момент выделен один растровый объект в текущем документе. Обработка групп объектов и множества растровых изображений не поддерживается.

После установки всех необходимых параметров и нажатия кнопки “Generate Preview” («Предпросмотр») программа начинает обработку выделенного растра. Результат работы отображается в создаваемом программой отдельном слое “Vectorized”. После завершения работы алгоритма векторизации пользователь может либо изменить настройки для получения более качественного конечного набора векторных объектов, либо нажать кнопки “OK” либо “Cancel”. В первом случае построенное векторное представление растра остается, во втором – слой “Vectorized” удаляется со всеми содержащимися в нем объектами.

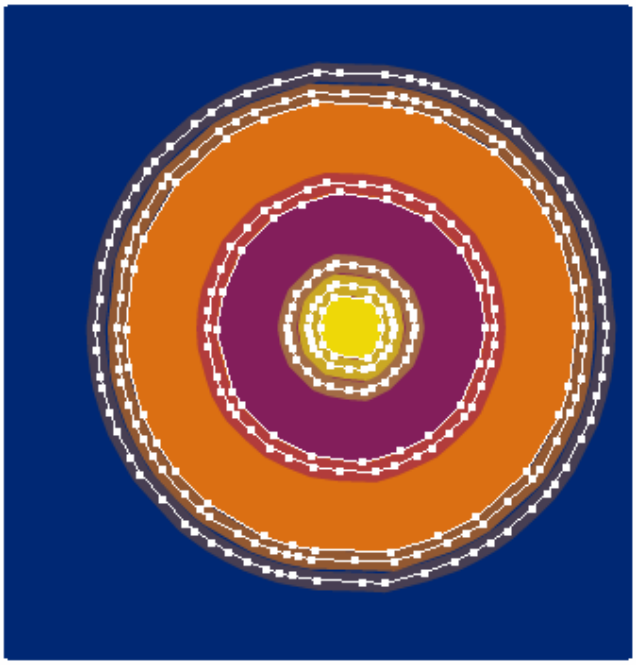
ПРИЛОЖЕНИЕ В. ПРИМЕРЫ РЕЗУЛЬТАТОВ РАБОТЫ МОДУЛЯ



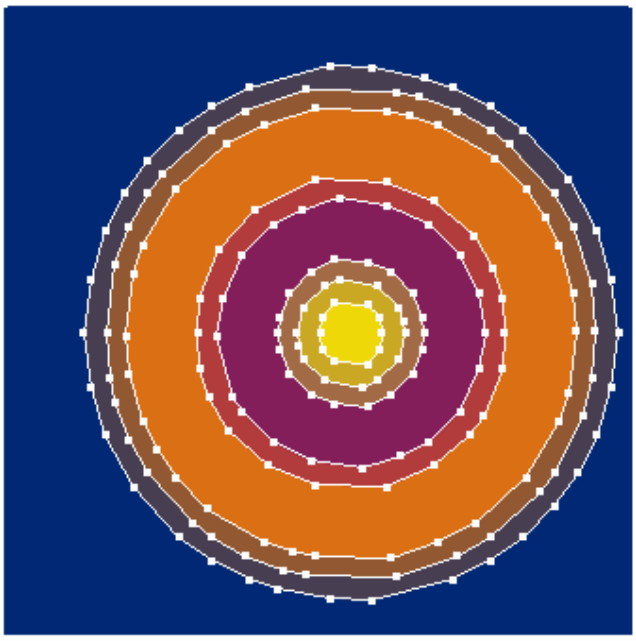
Сверху – построенная векторная модель, снизу – оригинал. Параметры: автоматическое определение типа – линия или полигон, сглаживание включено.



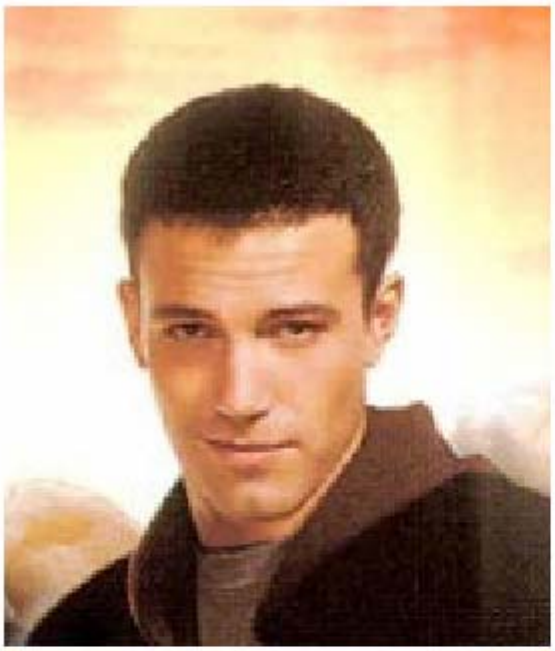
Оригинальное изображение – цветная градиентная заливка из центра по кругу



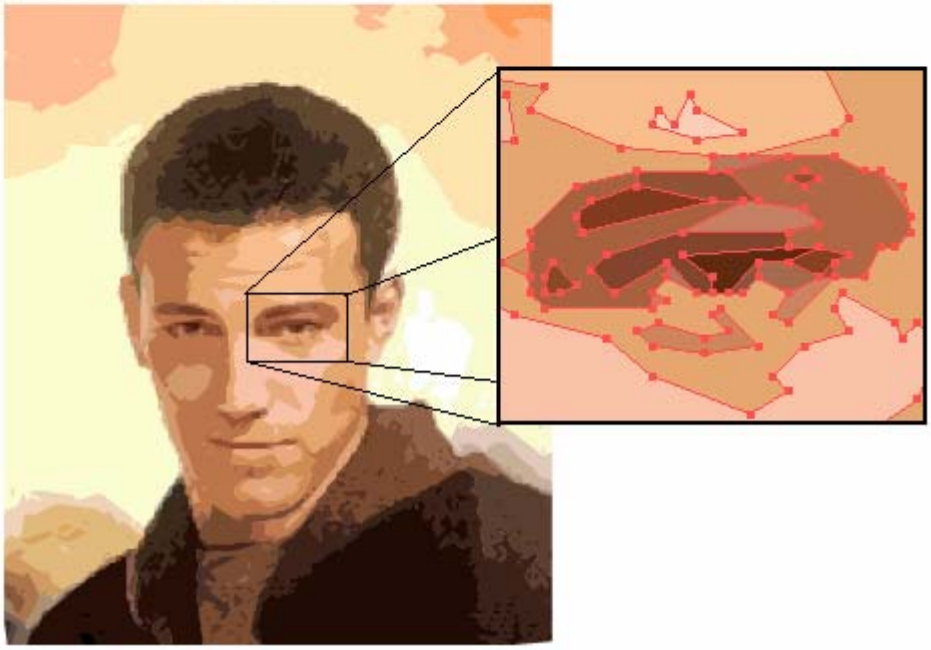
Векторизация с использованием как линий, так и полигонов. Сглаживание отключено.



Векторизация с использованием только линий. Сглаживание отключено.



Оригинал, ниже - результат векторизации полноцветного фотографического изображения. Сглаживание отключено, используются только площадные объекты.



При использовании автоматического выделения линейных и площадных объектов на фотографических изображениях получаются достаточно непривычные изображения, из-за того, что линия не совпадает точно с той областью цвета, на основе которой она создавалась. Поэтому получаются просветы, между векторными объектами, которые отчетливо видны на втором изображении. Т.е. для использования этих результатов необходимо еще дополнительно обработать получившиеся векторные объекты, в зависимости от того, какой эффект хотелось бы получить.

.

ПРИЛОЖЕНИЕ С. ДИСКЕТА С ИСХОДНЫМИ ТЕКСТАМИ