УДК 004.924

В.В. Карабчевский, С.Н. Магдалина
Донецкий национальный технический университет
karabch@pmi.dgtu.donetsk.ua
serezael@mail.ru

Реализация алгоритмического подхода к вопросу обработки изображений

В статье рассматривается интерпретатор скриптов, который может использоваться как часть более крупной системы для расширения ее возможностей. Интерпретатор разработан для графического редактора растрово-векторного типа для упрощения процесса генерации изображений, однако, может быть интегрирован и в другие системы.

Интерпретатор скриптов, графический редактор растрово-векторного типа, генерация изображений

Введение

В настоящее время наблюдается интенсивный рост вычислительной мощности различных цифровых устройств, включая персональные компьютеры. Это приводит к тому, что в системах обработки информации в реальном времени становится доступным больший объем вычислений без видимого увеличения временных затрат на них. С другой стороны, наблюдается рост мощности видеокарт, что позволяет использовать качественно новые методы работы с информацией. Использование мощных видеокарт позволяет добиться впечатляющих графических эффектов и разрабатывать красочные интерфейсы для различных устройств и систем визуализации. В то же время, освобожденная мощность процессора позволяет проводить более сложные "полезные" вычисления в программе.

Эти факторы позволяют по-новому взглянуть на решение различных проблем, в том числе проблем обработки изображений [1].

Анализ существующих методов

На сегодняшний день наиболее известными и яркими классами методов обработки и хранения графической информации являются растровая и векторная графика.

В растровой графике обрабатываемый объект представляет собой матрицу пикселей (растр), цвет которых и формирует конечный вид изображения. Обработка этой матрицы заключается в воздействии на некоторые группы пикселей с целью изменения их цвета.

Растровая графика предоставляет достаточно точные методы для обработки изображений [2]. Имея возможность обработки отдельных пикселей, в результате работы можно добиться достаточно близких к желаемым результатов. К сожалению, прямая обработка растра изображения процесс длительный и утомительный, поэтому создание и обработка растровых изображений может занять длительное время и потребовать много труда. Также из-за непосредственного хранения матриц пикселей, обработка и хранение растровых изображений требует большого объема памяти.

Наиболее выдающимся растровым графическим редактором является Adobe Photoshop. Он предоставляет мощные средства для прямого рисования на изображении, а также изменения тоновых и цветовых характеристик изображения в целом [3]. К недостаткам программы можно отнести невозможность математического описания обрабатываемых объектов.

Вторым широким классом методов графической информации является векторная графика. В векторной графике объекты хранятся в памяти в виде их математического описания [4]. Их вывод (чаще всего, на экран компьютера) производятся путем вычисления растрового образа объекта на основе его свойств – формул, задающих геометрическую форму, видов заливки и контура, цветов. Такой тип управления объектами позволяет легко ими манипулировать – изменение лишь одного из свойств может сильно повлиять на результирующий вид объекта. С другой стороны, конечный вид векторных объектов нельзя изменить напрямую – хоть векторный объект в конечном итоге и преобразуется в группу пикселей при выводе на экран, изменить его нельзя. Это приводит к тому, что изображение, состоящее из векторных объектов, как правило, выглядит достаточно искусственно.

Векторные объекты занимают очень маленький объем в памяти компьютера, тем не менее, требуют больших вычислительных затрат при построении.

Одним из наиболее известных векторных редакторов является Corel Draw [5]. Он имеет множество инструментов для создания векторных объектов и наложения на них разнообразных эффектов. Corel Draw замечательно справляется с

задачей создания различных плакатов, логотипов, знаков и других объектов, требующих частого изменения свойств и размеров.

Как можно заметить, каждый из представленных методов имеет свои преимущества и недостатки. В связи с этим, актуальной является проблема разработки методов обработки графики, которые бы сочетали в себе достоинства этих методов и избегали бы их недостатков.

Постановка задачи

Целью проведения данного исследования является разработка новых методов хранения и обработки графической информации, которые бы упростили создание и обработку изображений.

Также полученные методы необходимо применить на практике в виде разработки приложения, представляющего собой графический редактор, обладающий заявленными выше свойствами. Кроме того, важным аспектом является адаптивность редактора и его возможность выполнять не только запрограммированные при его разработке команды, но и пользовательские сценарии. Для реализации этой возможности разработан специальный интерпретатор скриптов, структура которого будет рассмотрена далее в этой статье.

Разработка концепции редактора

Первоочередной задачей при разработке редактора является определение общего способа хранения данных. Для ее решения была выбрана классическая слоистая модель документа, предполагающая, что обрабатываемый документ состоит из множества слоев, которые поочередно выводятся на рабочую область документа, которая, в свою очередь, выводится на экран. При этом, все видимые объекты документа располагаются на этих слоях и выводятся на соответствующий слой перед его выводом. Такая организация данных позволяет разбить объекты системы по смысловым категориям (слоям), а также упрощает работу с ними благодаря тому, что каждый слой можно сделать невидимым, переместить в стеке слоев вверх или вниз или объединить с другим. Кроме того, сами слои имеют свою собственную, напрямую редактируемую, растровую область, размер которой равен размеру документа. Схематично слоистая модель документа представлена на рисунке 1.

Здесь :

- a – рабочая область документа,
- b – растровая область слоя 1,
- c – выделенный фрагмент растровой области слоя 1,
- d составной объект, расположенный на слое 2,
- e – объекты, входящие в состав составного объекта d.

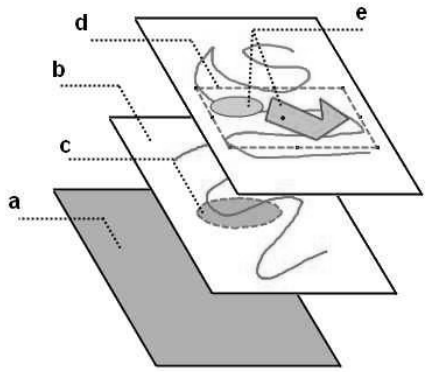

Рисунок 1 – Слоистая модель документа.

Цветовая модель

Кроме определения порядка вывода объектов в документе, также необходимо выбрать цветовую модель, используемую в редакторе. Цветовая модель определяет набор операций, которые можно выполнять над цветами. Наиболее распространенной, понятной и естественной для отображения на экране является цветовая модель RGBA, в которой результирующий цвет пикселя получается путем смешивания трех его основных компонент – красного, зеленого и синего цветов. Кроме них, в цвете также присутствует компонент альфа-канала, или канала прозрачности. Его значение указывает на то, в какой степени цвет пикселя будет смешан с цветом пикселя, на который выводится данный пиксель. Математически этот процесс описывается следующим образом.

Пусть пиксель цвета (Rf,Gf,Bf,Af) накладывается на пиксель цвета (Rb,Gb,Bb,Ab). Тогда цвет результирующего пикселя будет вычислен по формуле :

```
A = max(Af,Ab) + (1 - max(Af,Ab)) * min(Af,Ab);
R = Af * Rf + (1 - Af) * Rb;
G = Af * Gf + (1 - Af) * Gb;
B = Af * Bf + (1 - Af) * Bb;
```

Эта формула соответствует логическим законам формирования цвета при наложении цветов.

Кроме смешивания, также важной цветовой операцией, используемой в редакторе, является умножение цвета на множитель. Выполняется это по формулам:

```
R2 = R1 * f;
G2 = G1 * f;
B2 = B1 * f;
A2 = A1 * f;
```

где

R1, B1, G1, A1 – исходный цвет,

$0 \le f \le 1$ – множитель,

(R2,B2,G2,A2) – результирующий цвет.

Также следует отметить, что наличие цвета в некотором пикселе выражается дробным числом, лежащим в интервале [0;1]. При умножении конкретного цвета на это число и будет получен цвет пикселя.

Растровая часть

Слоистая модель документа подразумевает наличие у каждого слоя растра – матрицы пикселей, представленных своими цветами – который можно изменять с помощью растровых инструментов редактора.

Одним из ключевых растровых объектов редактора является объект "маска" - матрица элементов, каждый из которых определяет, в какой доли следует смешать цвета в данном пикселе матрицы, чтобы получить результирующий цвет. Маска имеет параметр количества цветов, который определяет мерность каждого ее элемента.

Математически каждый элемент маски может быть описан следующим образом.

Пусть N – мерность (количество цветов) маски. Также пусть Ci – доля i-ого цвета в пикселе маски (i = 0..N-1).

Тогда для каждого элемента маски должны выполняться два неравенства :

$$\sum_{i=0}^{N-1} Ci \le 1$$

$$\forall Ci : 0 \le Ci \le 1 \quad (i=0..N-1)$$

То есть доля каждого цвета в элементе не может превышать 1, и сумма долей всех цветов элемента не может превышать 1.

Таким образом, маска описывает закон, по которому формируется цвет пикселей в заданной растровой области. Тем не менее, это лишь общее правило определения цвета растра. Чтобы это правило применить, необходимо применить к маске конкретные цвета, а саму маску – к конкретной области растра. Для этого предназначен объект "кисть".

Кисть по своей сути является реализацией маски. Она имеет некоторую ширину и длину в пикселях, возможно, но не обязательно, отличную от ширины примененной к кисти маски. В случае несовпадения размеров, матрица маски будет приведена к размерам кисти методом бикубической интерполяции. Это позволит применить маску к кисти без особенных потерь качества при преобразовании.

Также кисть имеет некоторое количество конкретных цветов, которое обязательно должно совпадать с количеством цветов маски, примененной к кисти.

Такая организация взаимодействия необходима для того, чтобы общие по сути форму и цвета маски можно было приводить к частным их проявлениям в документе.

Для примера, применив обычную маску, представляющую собой круг радиуса R (рис. 2.a), к кисти, размеры которой составляют (x1 ; y1), такие, что x1 > y1, получим эллиптическую кисть, изображенную на рисунке 2.b. Изменив размеры кисти на (x2 ; y2), такие, что x2 < y2, получим кисть, форма которой представлена на рисунке 2.c.

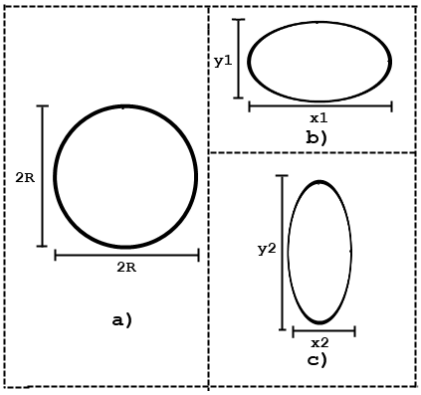

Рисунок 2 – a) – исходная маска, b) – вид кисти 1, c) – вид кисти 2.

Как можно увидеть, такой подход позволяет легко манипулировать формой и цветовым содержанием кисти (изменяя цвета, можно также легко влиять на внешний вид кисти с помощью маски).

Однако кисть имеет не только сформированный на основе маски растровый образ, а также и свой собственный экземпляр маски, полученный подгонкой исходной маски под размеры кисти. Хранение измененной маски необходимо для быстрого пересчета образа кисти при смене цветов, а также для выполнения операции рисования.

Прямое рисование кистью

Благодаря маске кисть располагает достаточным потенциалом для достижения многих эффектов. Для их реализации необходимо ввести еще один объект – "мазок". Мазок представляет собой маску, размеры которой совпадают с размерами документа. В обычном состоянии эта маска пуста (все элементы равны нулю). Однако, при рисовании в месте, затронутом кистью, она заполняется значениями элементов маски кисти. Значения элементов маски мазка переопределяются соответствующими значениями маски кисти. После окончания операции рисования на основе маски мазка и текущих цветов кисти формируется

его растровый образ, который выводится на текущий слой.

Такой подход позволяет реализовать режимы рисования, порождающие достаточно интересные эффекты. Одним из таких режимов является режим приоритетов.

В этом режиме каждому цвету маски кисти ставится в соответствие значение приоритета, равное его порядковому номеру в маске. При этом наиболее приоритетными цветами считаются цвета с меньшим номером. То есть C0 приоритетнее C2. При рисовании в режиме приоритетов происходит не переопределение каждого элемента маски мазка, а изменение его таким образом, чтобы значение доли более приоритетных цветов было максимальным (при сохранении основного условия маски – сумма долей не больше 1). То есть если в элементе маски мазка значение более приоритетного цвета меньше, чем в соответствующем элементе маски кисти, то значение доли этого цвета в элементе маски мазка переопределяется, при этом возможно уменьшение долей менее приоритетных цветов так, чтобы сумма по-прежнему не была больше 1.

Алгоритмически этот процесс представлен на рисунке 3.

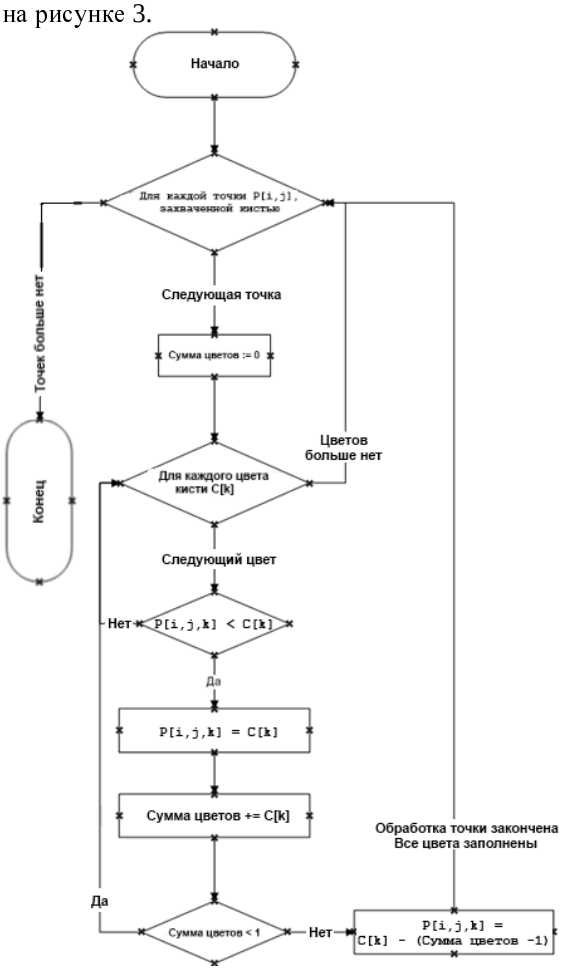

Рисунок 3 – Блок-схема алгоритма рисования кистью в режиме приоритетов.

Пример рисования кистью в режиме приоритетов будет представлен далее в разделе описания реализации редактора.

Векторная часть

Если растровые инструменты в основном нужны для правки готовых изображений, разных коррекций и т.п., то векторные объекты нужны для упрощения генерации изображений. Задав какие-то свойства объектам, получим некоторое сгенерированное изображение. Или быстро изменим готовое.

Все объекты независимо от класса имеют общие свойства – битовый образ, поворот и размер объекта.

Битовый образ объекта представляет собой растр, размер которого равен размеру объекта. Этот растр является видом объекта в пространстве документа и при выводе объекта на экран выводится именно его битовый образ. Битовый образ объекта генерируется всякий раз, когда получено достаточно информации для вывода объекта на экран.

Поворот объекта – это угол поворота объекта вокруг его центральной точки. Угол поворота объекта не влияет на геометрические свойства объекта и лишь изменяет его размер и битовый образ, тем самым позволяя изменять конечный вид объекта в документе.

Размер объекта по горизонтали и вертикали – это расстояние в пикселях от самой левой его точки до самой правой, и, соответственно, от самой верхней до самой нижней. Размер объекта определяет границы пространства, охватываемого объектом в документе, а также размер его битового образа. При этом размер объекта зависит как от специализированных свойств класса объекта, так и от угла его поворота.

Описанные понятия можно продемонстрировать на примере.

Возьмем объект класса "прямоугольник" со сторонами 100 пикселей (т.е. квадрат). Его битовый образ представляет собой прямоугольник размером 100 на 100 пикселей, угол поворота – 0 градусов, размеры – также 100 на 100 пикселей, а битовый образ представляет собой сплошной квадрат (см. рис 4.a).

Если этот квадрат повернуть на 45 градусов вокруг его центра (рисунок 4.b), то его размер станет равным 142x142 точки – именно таков размер диагоналей квадрата, и соответственно протяженность полученной фигуры слева направо и сверху вниз. Битовый образ фигуры также станет размером 142x142 точки. При этом области a,b,c,d в битовом образе будут прозрачными.

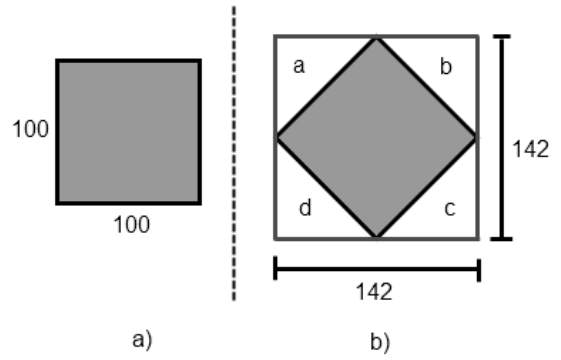

Рисунок 4 – Пример изменения базовых свойств объекта.
а) – начальный образ объекта
б) – объект, повернутый на 45 градусов

Все векторные объекты в редакторе делятся на несколько больших классов, каждый из которых, кроме общих, имеет свои собственные уникальные свойства и операции.

Первым классом объектов является простой растровый объект. По сути, такой объект является уменьшенной копией слоя – он имеет в своем составе лишь битовый образ, который можно редактировать растровыми инструментами. Объекты этого класса удобны в нанесении различного рода масок, текстур и прочих изменяющих внешний вид других объектов элементов.

Второй класс объектов – простая геометрическая форма. Этот класс имеет несколько подклассов, таких, как прямоугольник, эллипс, многоугольник и другие фигуры, задание которых требует небольшого количества данных.

Третий класс – кривые Безье. Если объекты второго класса удобно задавать прямым вводом данных, то для кривых Безье наиболее подходящим является визуальный способ задания по опорным точкам [6]. Кривые Безье позволяют с легко создавать кривые практически любой формы, что делает их идеальным выбором при создании различных контуров, путей и направляющих.

Объекты базовых классов полезны при генерации изображения сами по себе, однако, куда полезнее могут оказаться группы объектов, или так называемые составные объекты.

Суть использования составных объектов в том, что несколько объектов группируются в один объект. При этом, их положение, размер и поворот задаются уже не в пространстве документа, а в пространстве составного объекта. Сам же составной объект имеет свои размеры, положение и поворот в пространстве документа или, если он также является частью составного объекта, этого составного объекта. Таким образом, появляется возможность манипулирования геометрическим расположением всех объектов сразу.

Составной объект представлен на рисунке 5.

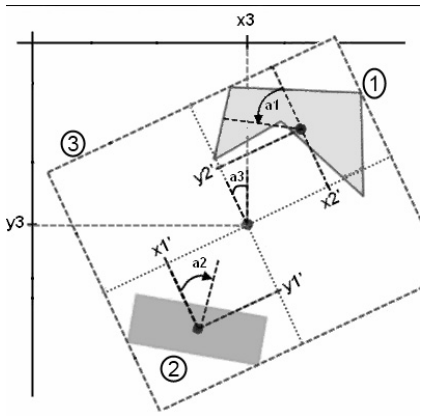

Рисунок 5 – Составной объект.

Здесь составной объект (3) состоит из двух простых объектов (1) и (2). При этом координаты объекта 3 (x3 ; y3) задаются в пространстве всего документа от его левого-верхнего угла.

Объект 3 повернут на угол a3, поэтому его оси координат (на рисунке показаны пунктирной линией) также повернуты на этот угол. Объекты 1 и 2 входят в состав объекта 3 и их координаты задаются от центра объекта 3, о чем можно судить по пунктирным перпендикулярам, идущим от центров объектов 1 и 2 к осям координат объекта 3. Сами объекты 1 и 2 в свою очередь повернут на углы a1 и (-a2) соответственно. При этом поворот осуществляется относительно центров объектов.

Интерпретатор скриптов и его интеграция с программой

В большинстве графических редакторов видна четкая направленность в сторону либо растровой обработки графики, либо векторной. Несмотря на то, что в разрабатываемом редакторе присутствуют инструменты для работы с обоими видами графики, процесс обработки и генерации изображений можно сделать еще более простым и многогранным.

При разработке крупных систем программисты стараются учесть все потребности потенциальных пользователей программы и оснащают ее набором необходимых инструментов. Однако, часто так бывает, что возможностей программы не хватает для удовлетворения потребностей пользователей. Существующими в программе средствами бывает сложно или даже невозможно осуществить цели пользователей. В таких ситуациях разработчикам приходится выпускать различные дополнения, патчи и новые версии продукта.

Однако, такая схема не обеспечивает приемлемой мобильности – сколько бы ни выходило дополнений к программе, потребности всех пользователей все равно не будут

удовлетворены. Кроме того, разработка дополнений лежит на плечах узкого круга разработчиков системы, что затрудняет выход новых версий системы.

В связи с этим, в рамках проекта по разработке графического редактора был разработан интерпретатор скриптов, который помог бы расширить базовые возможности графического редактора, а также был бы полезен при разработке других систем.

Интерпретатор скриптов KShScript представляет собой библиотеку, написанную на языке C++ в среде Visual Studio 2010.

Интерпретатор предназначен для того, чтобы дать возможность конечным пользователям программы влиять на ее возможности без изменения исходного кода программы. Однако, пользователь сможет воспользоваться возможностями интерпретатора только если программист, который пишет программу, интегрирует библиотеку в программу. Распределение ролей программиста и пользователя при работе с интерпретатором показано на рисунке 6.

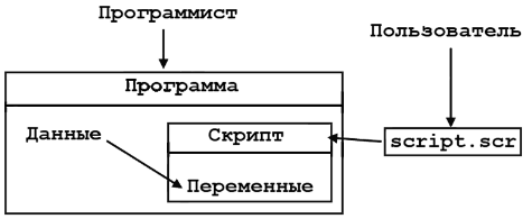

Рисунок 6 – Распределение ролей при работе с интерпретатором.

Скрипты могут вызываться разными методами, однако, наиболее логично применять для этого событийную модель.

Программист пишет программу, в которой наступают те или иные события. Пользователь, в свою очередь, пишет скрипты для интерпретатора скриптов, который встроен в программу, и связывает эти скрипты с событиями. После этого каждый раз при наступлении того или иного события, будет выполняться связанный с ним скрипт, который будет воздействовать на данные программы.

Структура скрипта и скриптовый язык

Пример текста скрипта представлен на рисунке 7. В данной части статьи будет описана его структура и операции, а в следующей – непосредственно работа.

Текст скрипта состоит из двух частей, разделенных ключевым словом PR_PROGRAM.

В первой части с помощью ключевого слова PR_VAR, объявляются используемые в скрипте переменные. Общий вид конструкции объявления переменной такой:

PR_VAR <вид> <тип> <имя>

```
PR_VAR A PT_ABYTE image
PR_VAR A PT_INT image_sizeX
PR_VAR A PT_INT image_sizeY
PR_VAR A PT_INT brush_sizeX
PR_VAR A PT_INT brush_sizeY
PR_VAR A PT_INT pointX
PR_VAR A PT_INT pointY
PR_VAR L PT_INT i
PR_VAR L PT_INT j
PR_VAR L PT_INT index
PR_PROGRAM
i = pointY - brush_sizeY
while (i < pointY + brush_sizeY)
{
   if ((i < image_sizeY) & (i > -1))
   {
       j = pointX - brush_sizeX
       while (j < pointX + brush_sizeY)
       {
           if ((j < image_sizeX) & (j > -1))
           {
               index = j + i * image_sizeX
               image[index] = 255 - image[index]
               index = index + 1
               image[index] = 255 - image[index]
               index = index + 1
               image[index] = 255 - image[index]
           }
           j = j + 4
       }
   }
   i = i + 1
}
PR_END
```

Рисунок 7 – Пример текста скрипта.

Переменные скрипта могут различаться по видам и типам.

По **видам** переменные скрипта делятся на локальные переменные и аргументы.

Локальные переменные – это служебные переменные скрипта. Перед началом выполнения скрипта под них происходит захват памяти и их значения имеют смысл только по время выполнения скрипта. Локальные переменные требуют явной инициализации в некотором месте скрипта до первого их использования. Для объявления переменной как локальной используется значение поля вида "L".

Аргументы скрипта – это переменные скрипта, под которых не захватывается память. Аргументы скрипта по своей сути являются указателями на соответствующие базовые типы основной программы (программы, в которую встроен интерпретатор). В связи с этим, аргументы скрипта уже имеют некоторые значения (являющиеся отражением значений переменных внешней программы) и всякое изменение значений аргументов в скрипте автоматически приводит к изменению значений переменных внешней программы. Объявление переменной в качестве аргумента требует указания значения поля "вид" как "A".

Переменные в скрипте организованы таким образом, чтобы иметь возможность оперировать с тремя базовыми **типами** внешней программы (написанной на языке C++) : int, byte и float. Разделение типов на int и byte необходимо для корректной работы с памятью при обработке аргументов.

Детальное рассмотрение объявления переменных различных типов выходит за рамки

данной статьи. Ограничимся лишь тем фактом, что интерпретатор скриптов позволяет работать с переменными трех вышеописанных типов, а также с массивами и матрицами этих элементов, как локальными, так и в качестве аргументов. Для локальных переменных массивов и матриц после определения типа через пробел указывается их размерность.

После описания всех свойств переменной следует ее **имя**, которое должно быть локальным внутри скрипта. Имя переменной скрипта не имеет никакого отношения к имени переменной внешней программы.

После описания всех переменных следует ключевое слово **PR_PROGRAM**, сообщающее интерпретатору, что начинается раздел инструкций скрипта.

Инструкции скрипта делятся на три вида.

Первый – присваивание.

Присваивание это инструкция, которая присваивает переменной или элементу массива или матрицы значение выражения, стоящее справа от знака "=". В качестве выражения служит математическое выражение, состоящее из переменных, элементов массивов или матриц, констант и функций, удовлетворяющее общим правилом математической записи выражений.

В выражениях допустимо использование скобок любого уровня вложенности, математических операторов "+", "-", "*", "/", а также встроенных в интерпретатор математических функций: синус, косинус, модуль и квадратный корень.

При вычислении значения выражения производятся все необходимые проверки на область допустимых значений операндов при выполнении операций. В случае возникновения недопустимых значений (например, отрицательного числа в подкоренном выражении или деления на ноль) интерпретатор завершит работу скрипта, выведя сообщение об ошибке в специальный файл лога.

Вторым типом инструкций являются условия.

Условия представляют собой классическую конструкцию if .. else, возможно, без части else. При этом проверяемое условие состоит из совокупности выражений, используемых для записи инструкций присваивания, или других условий. В процессе проверки условия производится вычисление всех вложенных выражений и условий. После этого над ними производятся логические операции: логическое-И, логическое-ИЛИ, логическое-НЕ, сравнение "меньше", сравнение "больше" и сравнение "равно". После проведения всех логических операций определяется – истинно ли вычисленное условие.

Третьим типом инструкций являются циклы.

Скрипты позволяют применять циклы "пока" классического вида – цикл выполняется, пока условие, заданное в нем, истинно. При этом условие задается точно так же, как и в случае с условным оператором.

После перечисления всех инструкций скрипта, следует ключевое слово **PR_END**, указывающее интерпретатору на конец текста скрипта.

Реализация редактора

Для применения разработанных методов работы с графикой был разработан графический редактор, главное окно которого представлено на рисунке 8.

Рисунок 8 – Главное окно редактора.

По умолчанию при запуске редактора доступны три окна – окно работы со слоями, палитра и окно свойств кисти. При запуске программы создается новый документ, содержащий один слой белого цвета.

При разработке редактора большое внимание уделялось не только его возможностям,

но и удобству работы с ним. Это выражается в том, что многие элементы интерфейса могут быть настроены под конкретного пользователя. Также множество функций системы могут быть вызваны с помощью горячих клавиш. Рассмотрим эти особенности редактора на примере окна палитры.

Окно палитры позволяет пользователю выбрать один или несколько рабочих цветов из общего списка цветов. для различных целей могут применяться различные палитры. При рисовании простых рисунков, как правило, используются палитры с небольшим количеством различных цветов, в то время как при рисовании сложных художественных элементов необходимо использовать палитру с большим количеством оттенков одного цвета.

Для настройки палитра имеет собственное окно свойств. В окне свойств палитры пользователь может настроить как внешний вид палитры, так и ее свойства, относящиеся непосредственно к функциям палитры. Выбранные параметры палитры можно сохранить в файл, чтобы затем загрузить их оттуда. Вместе с параметрами, представленными в окне свойств, сохраняются также цвета палитры, поэтому пользователь получает возможность создавать целые наборы палитр для различных целей.

Другой важной особенностью палитры, отражающей тенденцию работы в редакторе в целом, является наличие, кроме двух рабочих цветов, восьми текущих цветов. Текущие цвета – это цвета палитры, которые часто используются пользователем, но в данный момент не нужны. Выделяет их среди остальных цветов палитры то, что к ним привязаны быстрые клавиши [7].

В целом, графический редактор разрабатывается таким образом, чтобы пользователь был максимально сосредоточен на работе, минимально переключаясь на действия, связанные с изменением параметров или интерфейса.

Другой важной особенностью реализации редактора является его безопасность. К сожалению, среди программных продуктов немало таких, работа в которых является небезопасной с точки зрения сохранности рабочих данных или даже самой системы [8]. Так, ввод неправильных данных или загрузка неверного формата файла часто ведет к краху приложения или даже перезагрузке операционной системы. Данный графический редактор разрабатывается таким образом, чтобы исключить всякую возможность потери данных или зависания приложения в результате неправильных действий пользователя. Для этого каждое действие пользователя строго проверяется на безопасность и в случае обнаружения небезопасных или недопустимых действий, в специальном инфоокне выводится сообщение пользователю, а затем выполнение опасного действия прерывается.

Еще одной особенностью реализации редактора является использование библиотеки Direct2D, с помощью которой осуществляется вывод графической информации на экран. Direct2D является частью DirectX [9] и была специально разработана компанией Microsoft для использования аппаратных средств видеокарт для вывода двумерной графики на экран.

Благодаря использованию Direct2D редактор имеет хорошее быстродействие и высокую частоту обновления кадров даже на сравнительно слабых компьютерах [10].

После запуска редактора пользователю доступно окно управления слоями, в котором он может добавить, удалить, переместить вверх или вниз по стеку слоев, смешать с другим или переименовать слой.

Также активированным является инструмент "Кисть", которым можно выполнять рисование. Параметры кисти (размеры кисти в пикселях, различные режимы рисования, маску кисти) можно изменить в окне управления кистью.

В редакторе предусмотрена работа с двумя наборами масок – стандартными, определенными в редакторе, и пользовательскими, которые могут быть загружены из файла. Окно масок позволяет переключаться между этими наборами, а также выполнять загрузку и сохранение пользовательских в файл. Также в окне выбора масок показывается базовая информация о маске – ее имя, количество цветов, размеры.

На рисунке 9 представлены примеры рисования градиентной квадратной кистью, на рис.10 – круглой кистью с градиентной заливкой.

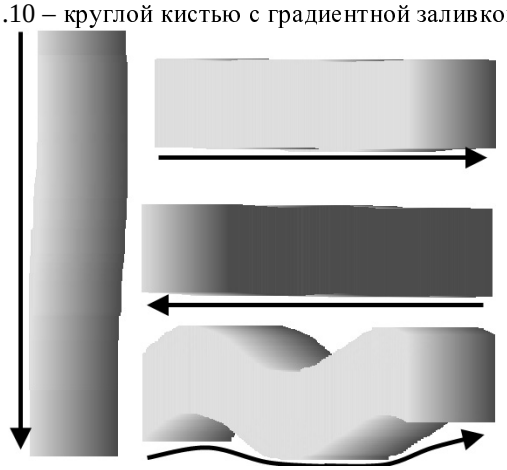

Рисунок 9 – Примеры рисования градиентной квадратной кистью.

Рисунок 10 – Пример рисования круглой кистью с градиентной заливкой

Однако, используя ту же маску, можно добиться тем же мазком совершенно другого эффекта, если включить режим приоритетов. Мазки, произведенные данной кистью с включенным режимом приоритетов, представлены на рисунке 11.

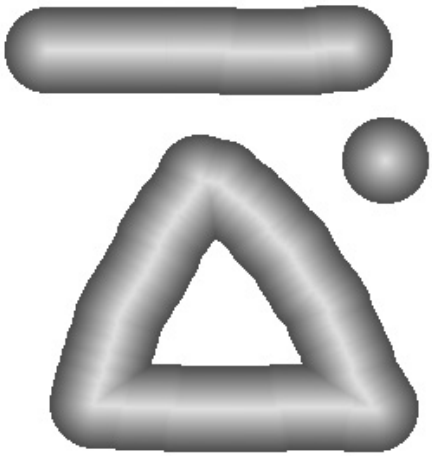

Рисунок 11 – Мазки, сделанные в режиме приоритетов.

Как можно заметить, режим приоритетов не нарушает целостности изначальной фигуры (образа кисти). При рисовании в этом режиме при правильном подборе маски можно легко достичь достаточно интересных эффектов.

Кроме вышеописанных методов рисования, в редакторе также реализована поддержка скриптов. В частности, с помощью пользовательского скрипта также можно производить рисование. Также с помощью скриптов можно создавать маски, которые в ручном режиме создать достаточно трудно.

Так, скрипт, текст которого представлен на рисунке 7, позволяет выполнять операцию инвертирования цветов в месте рисования кистью. Чтобы его загрузить, необходимо нажать кнопку "Скрипт" в окне свойств кисти и выбрать нужный скрипт. После того, как скрипт выбран, он будет загружен в систему и его аргументы будут связаны с данными редактора. В частности, ему будут переданы значения размеров кисти, размеров документа, растр текущего слоя и точка рисования. При рисовании этот скрипт будет инвертировать цвета тех точек растра, которые попали в область рисования кистью. Пример рисования кистью с загруженным скриптом инвертирования цвета представлен на рисунке 12.

На рисунке 13 представлен пример генерации при помощи скрипта многоцветной градиентной маски.

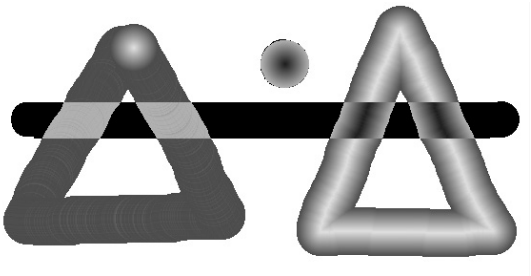

Рисунок 12 – Пример рисования кистью с загруженным скриптом инвертирования цветов.

Рисунок 13 – Пример генерации маски при помощи скрипта.

Выводы и перспективы развития

В результате работы были проанализированы основные методы обработки графической информации, выделены их достоинства и недостатки. На основании произведенного исследования была разработана концепция графического редактора смешанного типа.

Разрабатываемый редактор предоставляет как возможности работы с графикой растровыми методами, так и с векторными. Для этого был разработан базовый набор объектов обоих типов, установлен порядок их взаимодействия и методы работы с ними.

Также в качестве альтернативы привычным методам работы с графикой, был разработан интерпретатор скриптов, который позволяет подключать к редактору и выполнять пользовательские скрипты, тем самым расширяя базовые возможности редактора и предоставляя возможность математического и алгоритмического описания операций по обработке объектов редактора.

В дальнейшем планируется доработка применяемых в редакторе методов всех трех направлений – растровом, векторном и алгоритмическом. Будут разработаны новые методы взаимодействия растровых и векторных объектов, которые позволили бы использовать все сильные стороны растровых и векторных методов при формировании изображения.

Также планируется модернизация интерпретатора скриптов – оптимизация процесса выполнения скриптов для достижения более высокого быстродействия, усовершенствование скриптового языка.

Список литературы

1. Карабчевский В.В. Визуальное создание двумерных текстур средствами DirectX 9.0c / В.В. Карабчевский, С.Н. Магдалина // Наукові праці Донецького національного технічного університету. Серія: Інформатика, кібернетика та обчислювальна техніка. – 2009. – Вип. 10 (153). – С. 167–171.

2. Тайц Александр Самоучитель Adobe Photoshop 7 / Александр Тайц, Александра Тайц. – СПб.: БХВ-Петербург, 2005. – 688 с.: ил.

3. Легейда Владимир Photoshop CS2. Настоящий самоучитель / Владимир Легейда. – Век +, Корона-принт, НТИ, 2006. - 528 с.:ил.

4. Миронов Д. Corel Draw 11: учебный курс / Д. Миронов. – СПб.: Питер, 2002. – 448 с.: ил.

5. Corel Press. Corel Draw 11. The official guide. McGraw-Hill Osborne Media. – 2003. - 827 с.:ил.

6. Роджерс Д. Математические основы машинной графики / Д. Роджерс, Дж. Адамс ; пер. с англ. – М.: Мир, 2001. – 604 с., ил.

7. Документация по MFC [Электронный ресурс]. Режим доступа: http://msdn.microsoft.com

8. Ховард М. Защищенный код / М. Ховард, Д. Лебланк ; пер. с англ. – 2-е изд., испр. – М.: Издательско-торговый дом "Русская Редакция", 2004. – 704 с.: ил.

9. Горнаков Станислав DirectX 9: уроки программирования на C++ / Станислав Горнаков. – СПб.: БХВ – Петербург, 2005. – 400 с.: ил.

10. Миллер Том Managed DirectX 9.0 с управляемым кодом. Программирование графики и игр / Том Миллер. – SAMS, 2003. - 432 с.: ил.

Надійшла до редколегії 01.10.2011

В.В. КАРАБЧЕВСЬКИЙ, С.Н. МАГДАЛІНА
Донецький національний технічний університет

Реалізація алгоритмічного підходу до питання обробки зображень

У статті розглядається інтерпретатор скриптів, який може використовуватися як частина більш великої системи для розширення її можливостей. Інтерпретатор розроблений для графічного редактора растрово-векторного типу для спрощення процесу генерації зображень, однак, може бути інтегрований і в інші системи

Інтерпретатор скриптів, графічний редактор растрово-векторного типа, генерація зображень

V. KARABCHEVSKY, S. MAGDALINA
Donetsk national technical university

The implementation of an algorithmic approach to image processing

The article deals with the script interpreter, which can be used as a part of a larger system to expand its capabilities. The interpreter is designed for the raster-vector type graphic editor to simplify the process of image generation, however, it can be integrated into other systems.

Script interpreter, raster-vector type graphic editor, image generation