

**ПРАКТИКУМ
ПО
ОСНОВАМ КОМПЬЮТЕРНОЙ ГРАФИКИ**

ОГЛАВЛЕНИЕ

Оглавление

ОГЛАВЛЕНИЕ	3
ВВЕДЕНИЕ.....	6
1. ЛАБОРАТОРНАЯ РАБОТА «ОСНОВЫ GIMP»	7
Основные термины GIMP	7
Слои.....	7
Каналы.....	8
Выделения.....	8
История правки	8
Основные приемы использования GIMP	8
Панель инструментов	9
Окно изображения.....	10
Диалоги и панели	12
Работа с файлами	13
Изменение масштаба и навигация по изображению	15
Рисование. Кисти.....	16
Отмена действий	18
Задание по лабораторной работе	18
2. ЛАБОРАТОРНАЯ РАБОТА «ФОТОМОНТАЖ»	20
Выделение областей.....	20
Работа со слоями	26
Текст в GIMP.....	28
Преобразование изображения в слое	29
Фотомонтаж	33
Задание по лабораторной работе	33
3. ЛАБОРАТОРНАЯ РАБОТА	34
Коррекция цвета.....	34
Цветовой баланс.....	34
Коррекция тона, освещенности, насыщенности	34
Тонирование	36
Яркость и контраст.....	37
Гистограмма изображения.....	37
Коррекция цветовых кривых	40
Фильтры.....	41
Задание по лабораторной работе	43
«Визуализация».....	45
4. ЛАБОРАТОРНАЯ РАБОТА	46

Основы Inkscape	46
Работа с документами	48
Фигуры.....	48
Задание по лабораторной работе	54
5. ЛАБОРАТОРНАЯ РАБОТА	55
«СОЗДАНИЕ ВЕКТОРНОГО ЛОГОТИПА»	55
Размещение текста вдоль контура	55
Выполнение логических операций над фигурами	55
Работа с узлами	57
Задание по лабораторной работе	58
Варианты заданий:	58
Сообщение WM_PAINT	62
Событие Paint	63
Объект Graphics для рисования	63
Методы и свойства класса Graphics.....	64
public void DrawLine(Pen, Point, Point); public void DrawLine(Pen, PointF	
PointF; public void DrawLine(Pen, int, int, int, int);.....	64
Pen p =new Pen(Brushes.Black,2);	64
public void DrawLines(Pen, Point[]); public void DrawLines(Pen, PointF[];	
.....	66
Pen pen =new Pen(Color.Black, 2);	66
Point[] points =new Point[50];	66
DrawPolygon(Pen, Point[]);	67
Задание по лабораторной работе	69
7. ЛАБОРАТОРНАЯ РАБОТА	70
Работа с таймером	70
Создание анимации	70
Движение по траектории	71
Задание по лабораторной работе	73
спирали.....	73
Лемнискаты Бернулли	74
8. ЛАБОРАТОРНАЯ РАБОТА	75
«РАБОТА С РАСТРОВЫМИ ИЗОБРАЖЕНИЯМИ»	75
Отображение графических файлов	75
Компоненты OpenFileDialog и SaveFileDialog.....	76
Простой графический редактор	76
Задание по лабораторной работе	80
9. ЛАБОРАТОРНАЯ РАБОТА	82
«ПРЕОБРАЗОВАНИЯ НА ПЛОСКОСТИ»	82
Простейшие преобразования на плоскости	82
Преобразование поворота и отражения	83

Однородные координаты	85
Комбинированные преобразования	87
Программная реализация	87
Задание по лабораторной работе	90
Варианты заданий	90
10. ЛАБОРАТОРНАЯ РАБОТА	92
«3D ПРЕОБРАЗОВАНИЯ И ПОЛУЧЕНИЕ ПРОЕКЦИЙ»	92
Правосторонняя система координат	92
Преобразования в пространстве	93
Получение проекций	95
Программная реализация	98
Задание по лабораторной работе	99
Варианты заданий	99
11. ЛАБОРАТОРНАЯ РАБОТА	101
«ПОСТРОЕНИЕ ТРЕХМЕРНЫХ СЦЕН НА XAML»	101
Система координат и размещение камеры	101
Освещение сцены	102
Задание объектов трехмерной сцены	103
Применение материалов к модели	104
Трехмерные преобразования	105
Пример описания простой трехмерной сцены	106
Задание по лабораторной работе	107
12. ЛАБОРАТОРНАЯ РАБОТА	108
«ТРЕХМЕРНЫЕ ПРЕОБРАЗОВАНИЯ В WPF»	108
Связь процедурного кода и объектов описанных в XAML	108
Трехмерные преобразования в процедурном коде	108
Применение 3D-преобразований к отдельным элементам 3D сцены	110
Создание анимации с помощью таймера	114
Задание по лабораторной работе	118

ВВЕДЕНИЕ

Если заглянуть в историю, то можно проследить, как с момента появления первых ЭВМ люди стремятся разнообразить способы общения человека и машины, приблизившись к уровню общения человека с человеком. Это общение было бы гораздо более ограниченным, если бы не использовало один из наиболее простых способов – язык изображений, образов. Сегодня графические изображения на экране монитора современного персонального компьютера стали для нас нормой, совершенно неотъемлемым атрибутом интерфейса. Спектр применения компьютерной графики, помимо средства интерфейса «человек-машина», чрезвычайно широк: от создания рекламных роликов, компьютерных мультфильмов и игр, кроя одежды, малых и монументальных форм дизайна, компьютерной живописи до визуализации результатов научных изысканий [10]. Можно с уверенностью сказать, что популярность Internet, и в частности WWW, во многом объясняется широким применением графики.

Рынок программного и аппаратного обеспечения компьютерной графики – один из самых динамичных. Об этом можно судить по объему литературы и числу сервисов Internet, посвященных так или иначе компьютерной графике.

Предметом данной работы является обширная область компьютерных наук, посвященная представлению данных в памяти ЭВМ в графической форме. Это самое общее определение, так как под данными можно понимать как непосредственно хранящееся в виде файла изображение в одном из графических форматов, так и протокол обмена командами между пользователем и ЭВМ (то, что мы называем графическим интерфейсом), и битовую последовательность, сформированную для вывода на экран или печатающее устройство. Методы и способы представления и манипуляции этим видом данных относятся к компетенции компьютерной графики.

В работе рассматриваются различные способы представления изображений в памяти ЭВМ, методы и алгоритмы растеризации и обработки растровых изображений, матричные преобразования на плоскости и в пространстве, методы и алгоритмы удаления скрытых линий и поверхностей. Кроме того, приводятся основы использования графической библиотеки OpenGL, а также описываются базовые аппаратные средства, используемые при работе с изображениями.

Программный код, приведенный в пособии, создан в MS Visual Studio 2010 на языке C#.

1.ЛАБОРАТОРНАЯ РАБОТА «ОСНОВЫ GIMP»

GIMP – многоплатформенное программное обеспечение для редактирования растровых изображений (GIMP – GNU Image Manipulation Program). Редактор GIMP пригоден для решения множества задач по изменению изображений, включая ретушь фотографий, объединение и создание изображений.

Одной из сильных сторон GIMP является его доступность из многих источников для многих операционных систем. GIMP входит в состав большинства дистрибутивов GNU/Linux. GIMP также доступен и для других операционных систем вроде Microsoft Windows™ или Mac OS X™ от Apple (Darwin). GIMP – свободное программное обеспечение, выпускаемое под лицензией GPL (General Public License). GPL предоставляет пользователям право доступа к исходному коду программ и право изменять его.

Будучи весьма мощным продуктом, GIMP способен стать незаменимым помощником в таких областях, как подготовка графики для Web-страниц и полиграфической продукции, оформление программ (рисование пиктограмм, заставок и т. п.), создание анимационных роликов, обработка кадров для видеофрагментов и построение текстур для трехмерной анимации. Очень полезна функция создания и обработки анимационных роликов, позволяющая накладывать анимацию на объект как текстуру и выполнять определенные финишные операции после рендеринга.

Основные термины GIMP

Изображения

Изображение – основной объект, с которым работает GIMP. Под словом изображение подразумевается один файл с расширением TIFF или JPEG. Можно отождествлять изображение и окно, которое его содержит, но это будет не совсем правильно: можно открыть несколько окон с одним и тем же изображением. В то же время нельзя открыть в одном окне более одного изображения, и нельзя работать с изображением без отображающего его окна.

Изображение в GIMP может быть достаточно сложным. Наиболее правильной аналогией будет не лист бумаги, а, скорее, книга, страницы которой называются слоями.

Слой

Если изображение подобно книге, то слой можно сравнить со страницей внутри книги. Простейшее изображение содержит только один слой

и, продолжая аналогию, является листом бумаги. Слои могут быть прозрачными и могут покрывать не все пространство изображения.

Каналы

В GIMP каналы являются наименьшей единицей подразделения стека слоев, из которых создается изображение. Каждый канал имеет тот же размер, что и слой, и состоит из тех же пикселей. Смысл этого значения зависит от типа канала, например, в цветовой модели RGB значение канала R означает количество красного цвета, добавляемого к другим цветам пикселей.

Выделения

Часто при работе возникает необходимость изменить только часть изображения. Для этого существует механизм выделения областей. В каждом изображении можно создать выделенную область, которая, как правило, отображается в виде движущейся пунктирной линии.

История правки

Ошибки при редактировании изображений неизбежны, однако вы почти всегда можете отменить свои действия: GIMP записывает историю действий пользователя, позволяя при необходимости вернуться на несколько шагов назад. Однако история занимает память, поэтому возможности отмены не безграничны.

Основные приемы использования GIMP

Для того, чтобы открыть программу GIMP в Windows, нужно нажать на кнопку пуск и выбрать в меню GIMP пункт GIMP.

На рис. 1.1 показано стандартное расположение окон GIMP. Элементами окон являются:

1. Панель инструментов, которая содержит кнопки для выбора инструментов выделения, рисования, трансформации изображения и т. д.
2. Параметры инструментов: под панелью инструментов прикреплен диалог Параметры инструментов, который отображает параметры выбранного инструмента (в данном случае это инструмент «Кадрирование»)
3. Окно изображения: каждое изображение в GIMP отображается в отдельном окне. Вы можете открыть одновременно достаточно большое количество изображений, столько, сколько позволяют системные ресурсы.
4. Диалоги **Слои/Каналы/Контуры/Отменить**: этот диалог отображает структуру слоев активного изображения и позволяет управлять ими.
5. Диалоги **Кисти/Текстуры/Градиенты**: панель, расположенная ниже диалога слоев, показывает диалоги управления кистями, текстурами и градиентами.

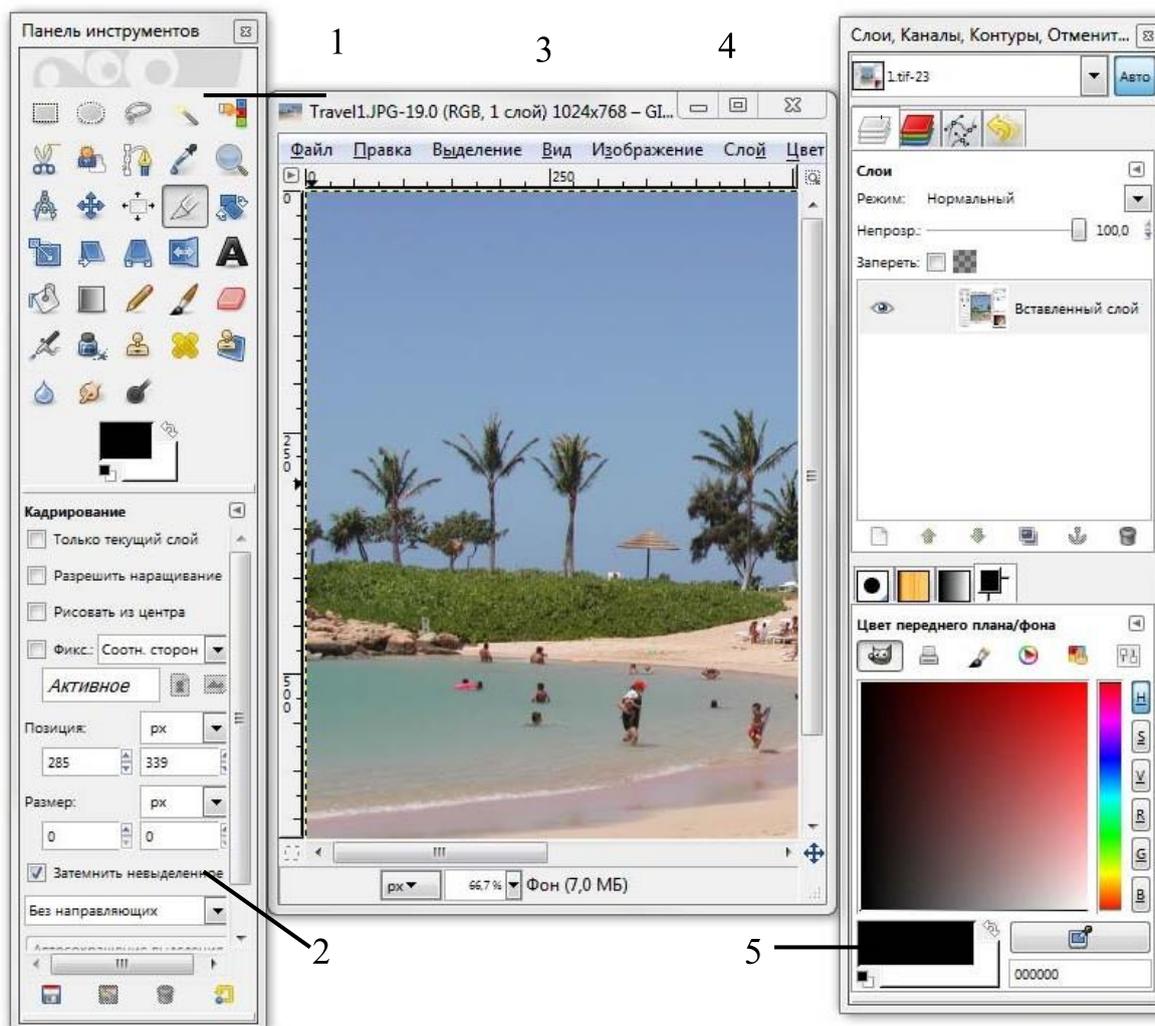


Рис. 1.1. Общий вид редактора GIMP

Приведенный набор – это минимальный набор окон. В GIMP используется более дюжины различных диалогов, которые можно открыть при необходимости. Опытные пользователи обычно держат открытыми панель инструментов (с параметрами инструментов) и диалог Слои.

Панель инструментов

Панель инструментов – единственная часть интерфейса программы, которую вы не можете продублировать или закрыть. Внешний вид Панели инструментов представлен на рис. 1.2.

1. Кнопки, которые активируют инструменты для разнообразных действий: выделение частей изображений, рисования, преобразования и т. п.
2. Цвета фона/переднего плана: область выбора цвета показывает текущий выбранный вами цвет переднего плана и фона, который

применяется во многих операциях. Щелчок по одному из них вызовет выборщик цветов, который позволяет вам установить другой цвет.

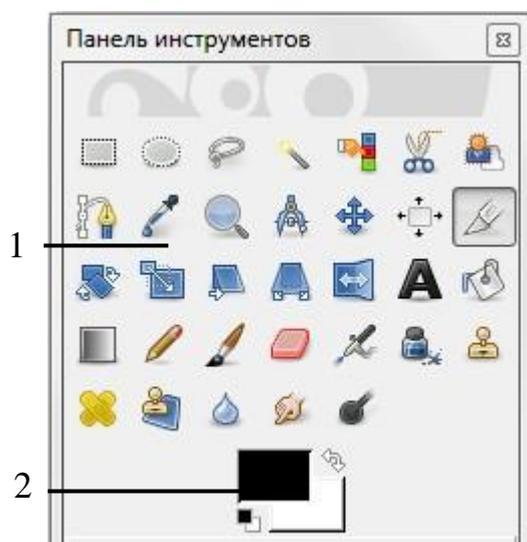


Рис. 1.2. Панель инструментов

Окно изображения

Каждое открытое изображение в GIMP отображается в своем собственном отдельном окне. Элементы окна показаны на рис. 1.3.

1. Заголовок изображения содержит ряд полезных сведений: имя файла изображения, наименование цветовой модели, номер текущего слоя, размер изображения в пикселях.
2. Прямо под заголовком находится меню изображения. С помощью этого меню вы можете получить доступ ко всем операциям, применимым к изображению. Вы также можете вызвать меню изображения щелчком правой кнопкой мыши на изображении, или щелчком левой кнопкой мыши по небольшому значку – «стрелке» в левом верхнем углу (3)
3. Щелчок по этой небольшой кнопке вызывает меню изображения, расположенное в столбец вместо строки. Такие кнопки широко используются в GIMP для вызова меню в различных окнах.
4. Линейки, которые используются для измерений. Если желаете, вы можете выбрать, в каких единицах измерения отображаются координаты. По умолчанию используются пиксели. Одно из основных действий для использования линеек – это создание направляющих. Если вы щелкните на линейке и перетащите на окно изображения, будет создана направляющая линия, которая поможет вам аккуратно располагать объекты на изображении.

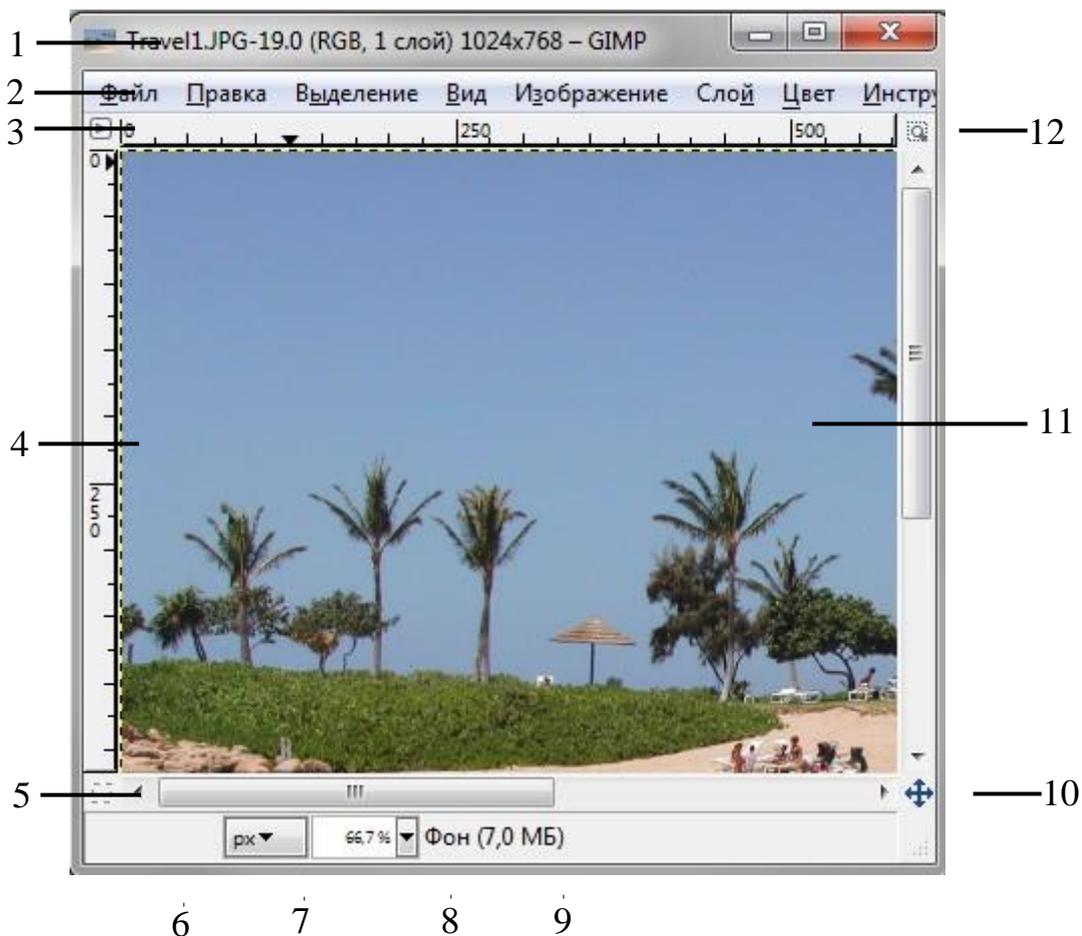


Рис. 1.3. Окно изображения

5. В левом нижнем углу окна изображения расположена небольшая кнопка, которая включает или выключает быструю маску, которая является альтернативным и часто полезным методом просмотра выделенной области внутри изображения.
6. В левом нижнем углу окна расположена прямоугольная область, используемая для отображения текущих координат указателя (положение мыши).
7. Используемыми по умолчанию единицами измерения для линеек и некоторых других целей являются пиксели. Вы можете заменить их на дюймы, сантиметры или другие единицы, доступные с помощью этого меню.
8. Меню изменения масштаба.
9. Область статуса расположена под изображением. Она отображает активный слой изображения, и количество занятой изображением системной памяти.
10. Панель навигации – небольшая кнопка крестовидной формы расположена справа внизу под изображением. Вы можете перемещать

щаться к другим частям изображения двигая мышь при нажатой кнопке.

11. Наиболее важная часть окна изображения это конечно, само изображение. Оно занимает центральную область окна и окружено желтой пунктирной линией, в отличие от нейтрального серого цвета фона.
12. Кнопка «Изменение размера изображения». На самом деле если эта кнопка нажата, при изменении размера окна будет меняться масштаб изображения.

Диалоги и панели

В GIMP версии 2.4 пользователь получил больше удобства в плане размещения диалоговых окон на экране. Вместо размещения каждого диалога в своем собственном окне, вы можете группировать их вместе с помощью панелей. Панель – это окно–контейнер, которое может содержать собрание постоянных диалогов, таких, как Параметры инструментов, Кисти, Палитры и др. Каждая панель имеет соединительные планки (рис. 1.4).

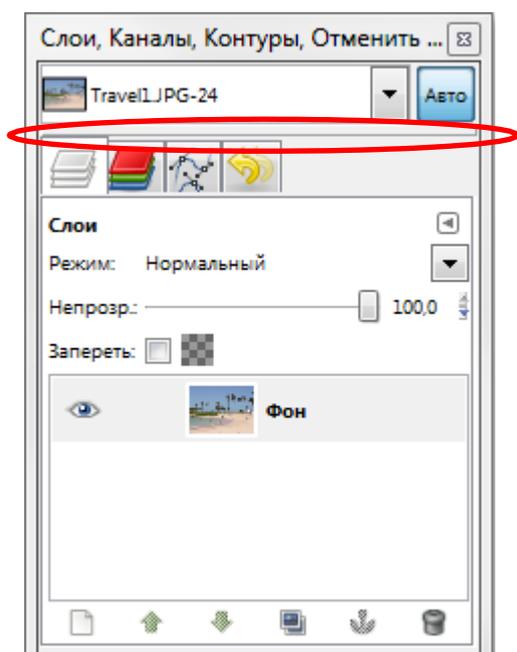


Рис. 1.4. Диалог с выделенной планкой

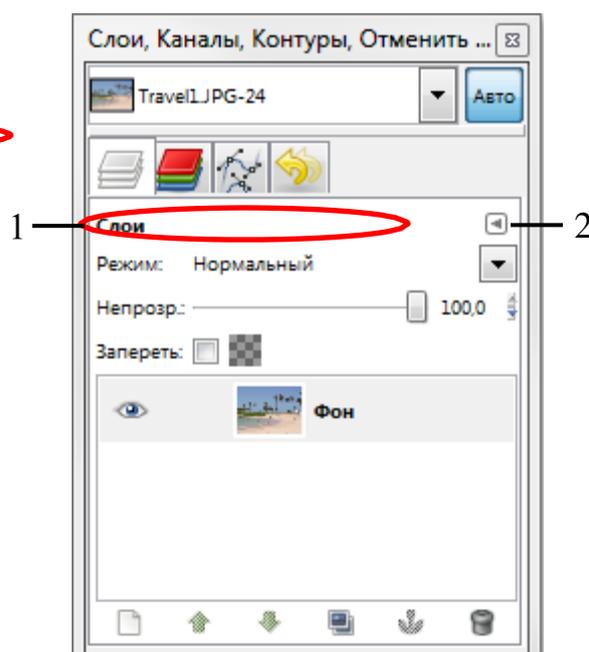


Рис. 1.5. Область перетаскивания диалогов

Каждая панель имеет область перетаскивания (рис. 1.5, область 1). При наведении указателя на область перетаскивания курсор изменит вид на форму ладони. Для присоединения диалога просто щелкните по области перетаскивания и перетащите его на одну из соединительных

планок в панели. Рис. 1.5 показывает область, позволяющую отделить диалог **Слои** от панели.

Вы можете перетащить более одного диалога в одну панель. Если хотите, они будут чередоваться в виде закладок, отображаемых в виде значков вверху диалога. Щелчок по закладке выдвигает диалог на передний план, следовательно, вы можете взаимодействовать с ним.

С помощью кнопки 2 (рис. 1.5) можно выполнить ряд действий с диалогами: добавление, закрытие, прикрепление, отсоединение вкладки.

Работа с файлами

Создание нового изображения

В GIMP вы можете создать новое изображение при помощи пункта меню: **Файл** → **Создать**. При этом откроется диалог «**Создать новое изображение**» (рис. 1.6), где можно установить начальные ширину и высоту файла.

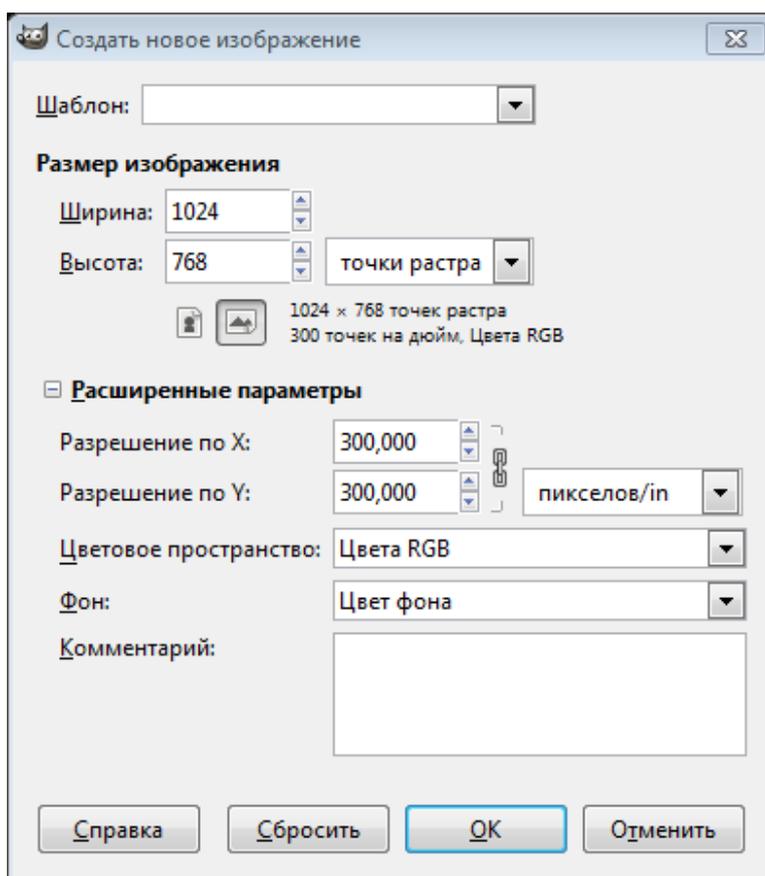


Рис. 1.6. Диалог «Создать новое изображение»

При выборе расширенных параметров устанавливается разрешение, цветовая модель и цвет фона.

Вторая команда главного меню **Файл** → **Создать**, позволяет создать изображение и вставить рисунок из буфера обмена. При этом будут установлены размеры изображения, которое находится в буфере обмена. Также при выборе этой команды возможен захват изображения с экрана, сканера или фотокамеры.

Открытие изображения

Доступно несколько способов открыть существующее изображение в GIMP. Наиболее очевидный – это открыть его с помощью меню **Файл** → **Открыть** в главном меню. При этом появится диалог выбора файла.

Другой способ заключается в использовании технологии drag&drop. Если значок файла перетащить на существующее изображение в GIMP, то файл добавится как новый слой или слои этого изображения.

Сохранение изображения

Для сохранения изображения необходимо выбрать команду **Файл** → **Сохранить**. После этого в появившемся окне (рис. 1.7) необходимо задать папку, куда будет сохраняться файл, имя и тип файла.

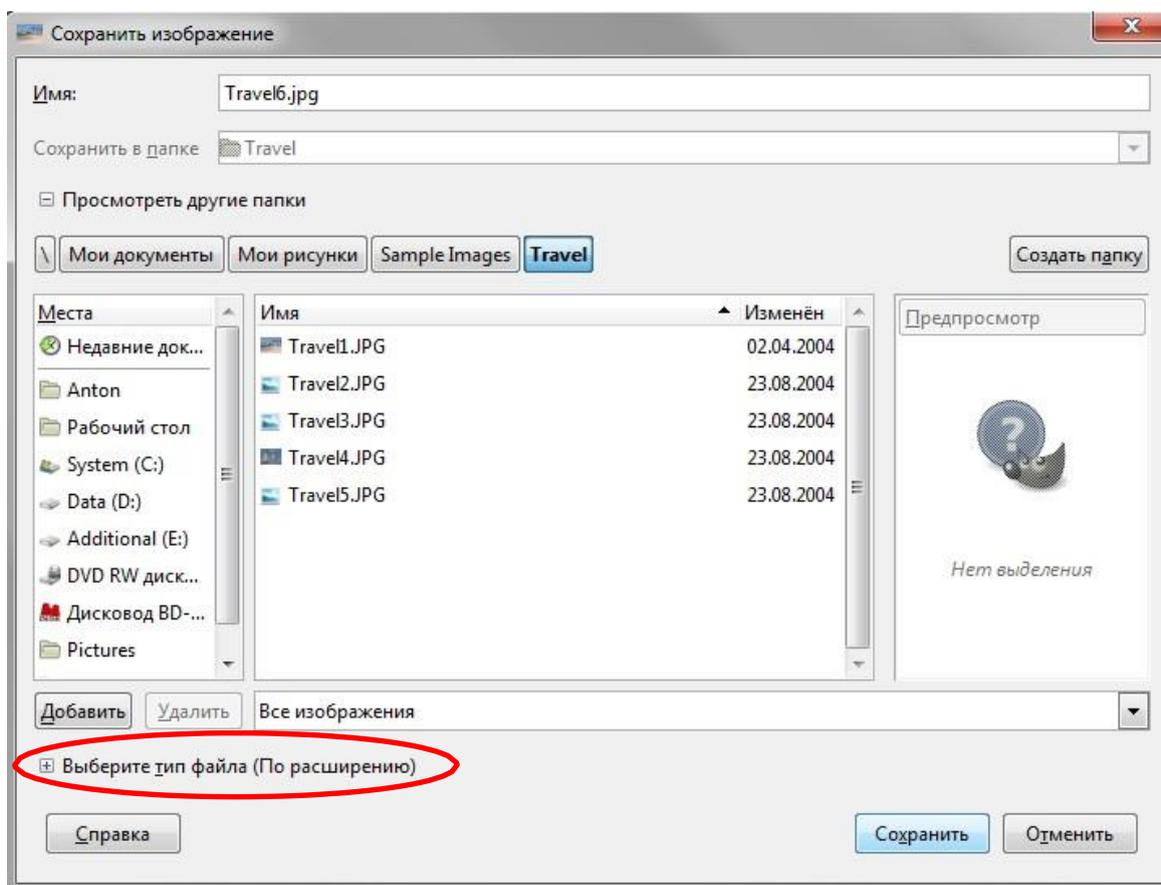


Рис. 1.7. Диалог сохранения файла

Для задания формата графического файла достаточно ввести соответствующее расширение (gif, bmp, tif и т. п.) после имени файла при выбранном параметре «По расширению», либо выбрать тип файла расширив диалог сохранения файла (рис. 1.8).

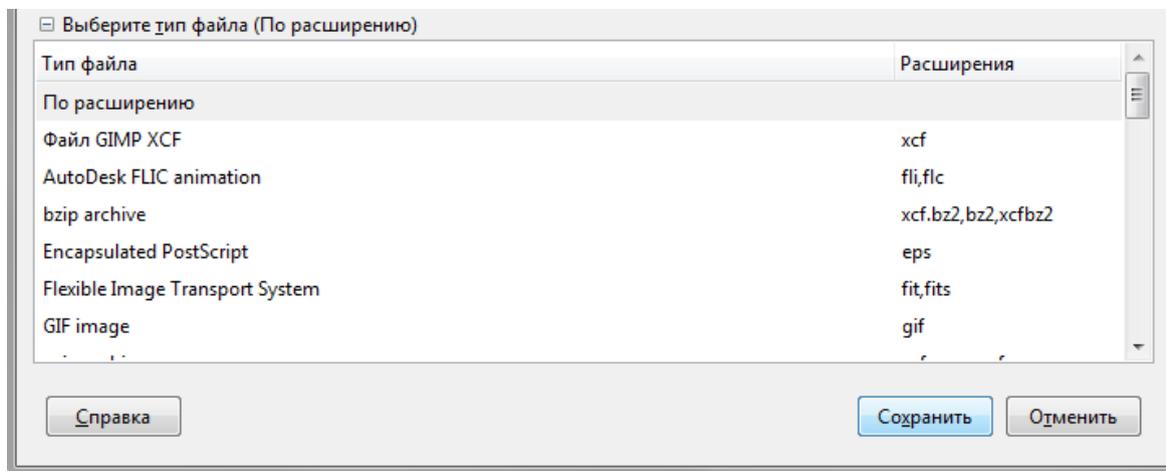


Рис. 1.8. Задание типа изображения

При сохранении изображения в некоторые форматы, могут появляться дополнительные окна для задания параметров изображения. Отметим формат JPG, при сохранении в котором можно задавать качество изображения. Чем выше будет задано качество, тем больший размер будет у файла, хранящего изображение.

Изменение масштаба и навигация по изображению

В ряде случаев, например, при обработке некоторых относительно маленьких областей, возникает необходимость изменения масштаба отображения изображения на экране. Это можно осуществить несколькими способами через интерфейсную часть программы, через клавиатуру и мышь. Текущий масштаб можно увидеть внизу окна изображения (рис. 3 область 8).

В меню изображения **Вид** найдите опцию **Масштаб**. Открывается подменю, в котором вы найдете множество возможностей изменить масштаб изображения на экране.

Способ изменения масштаба через клавиатуру заключается в использовании кнопок + (плюс) и – (минус).

Рассмотрим перемещение по увеличенному изображению через кнопку навигации.

1. Увеличим изображение до 400 %.
2. Нажмем на кнопку навигации (рис. 1.3 кнопка 10) и, не отпуская левую кнопку мышки, переместимся в любую часть масштабированного изображения.

Рисование. Кисти

Инструменты рисования представлены на рис. 1.9.



Рис. 1.9. Инструменты рисования

Инструменты Заливка, Карандаш, Кисть, Ластик, Аэрограф, Перо, Размывание/резкость, Палец, Осветление/Затемнение. Работа с этими инструментами отражена в их названии. Для простых действий применение данных инструментов не представляет сложности.

При выборе любого инструмента внизу панели инструментов отображаются его параметры (рис. 1.1 Область 2).

Основным инструментом рисования является кисть. При выборе кисти устанавливается **Режим**, который по умолчанию стоит в значении **Нормальный**. Это позволяет рисовать линии определённым цветом. Все остальные режимы при нанесении цвета учитывают также цвет фона, тем самым получается смешение цветов.

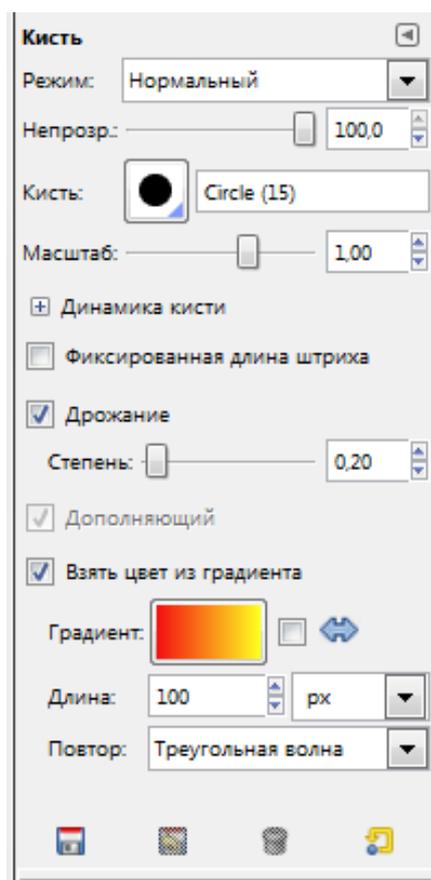


Рис. 1.10. Параметры кисти

Далее определяется на сколько будет непрозрачен цвет наносимый кистью, форма и размер кисти. Интересен тот факт, что любой выделенный объект, помещенный в буфер обмена командой **Правка** → **Копировать**, отображается в списке доступных форм кистей и может быть использован как кисть.

Ниже можно задать ряд параметров позволяющих добиться ряда специальных эффектов для кисти. Главное не забыть выбрать цвет, которым будем рисовать. Для выбора цвета на панели инструментов существуют специальные элементы (рис. 1.11).



Рис. 1.11. Элемент выбора цветов

Для выбора цвета кисти, карандаша, заливки используется область 1 на рис. 1.11. Для выбора цвета фона, цвета ластика используется область 2. Обе области используются для задания градиента. Градиент это плавный переход от одного цвета к другому. Элемент 3 используется для задания цветов по умолчанию: черного-основного и белого-цвет фона. Элемент 4 используйте для того, что бы поменять цвет фона с основным цветом.

При нажатии на область 1 или 2 (рис. 1.11) открывается дополнительная панель для выбора цвета (рис. 1.12).

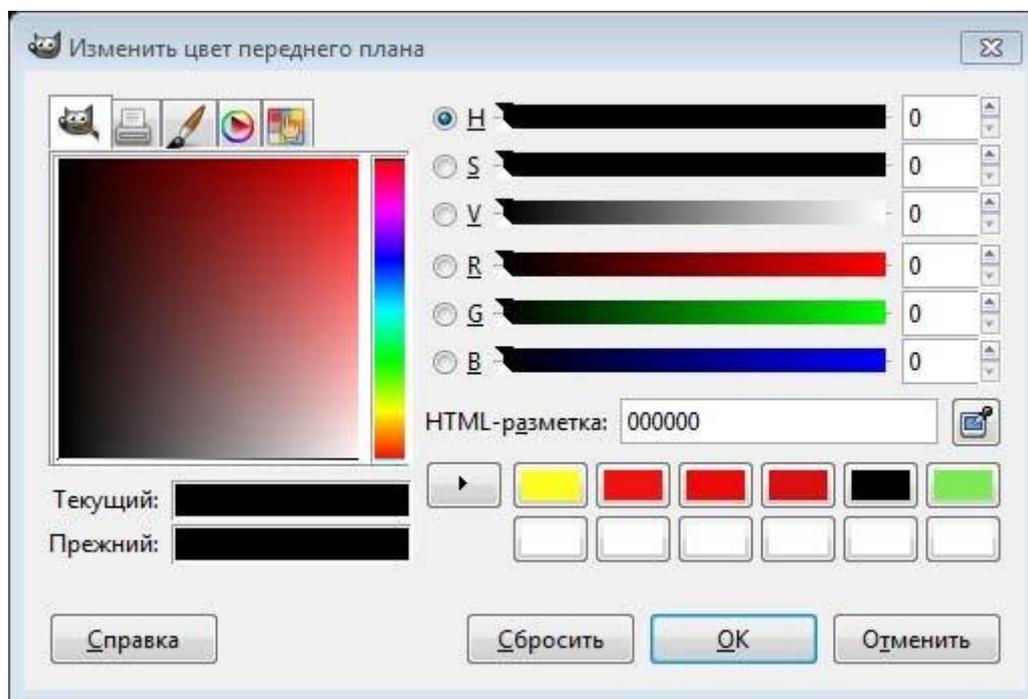


Рис. 1.12. Панель для выбора цвета

Отмена действий

Почти все, что делается с изображением, может быть отменено. Вы можете отменить последнее действие, выбрав в меню изображения **Правка** → **Отменить**, но эта операция применяется так часто, что вам лучше запомнить сочетание клавиш **Ctrl+Z**.

Сама отмена также может быть отменена. После отмены действия вы можете вернуть его, выбрав в меню изображения пункт **Правка** → **Повторить** или с использованием клавиши быстрого доступа **Ctrl+Y**. Часто это полезно при оценке эффекта какого-либо действия, с помощью его неоднократной отмены и повтора.

Если вы часто используете отмену и возврат на множество шагов за раз, возможно будет более удобно работать с диалогом **Истории действий** – прикрепляемой панелью, которая показывает небольшие эскизы каждой точки в истории отмены, позволяя вам перемещаться назад или вперед к точке, по которой вы щелкаете (рис. 1.13).

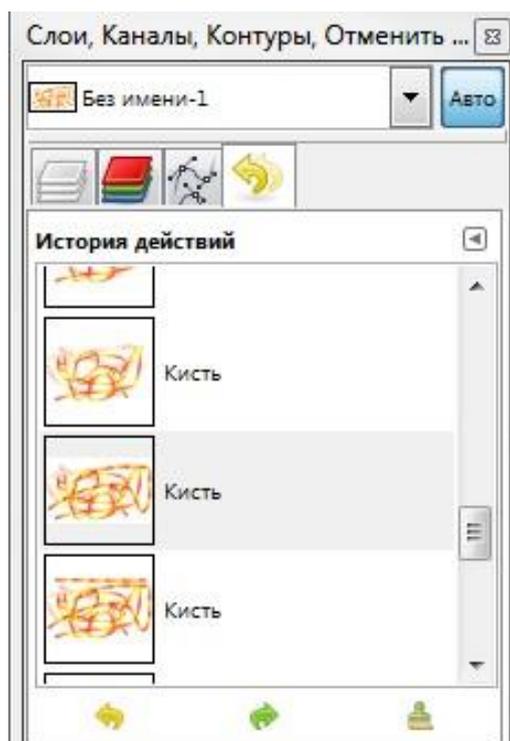


Рис. 1.13. Панель «История действий»

Задание по лабораторной работе

1. Запустите GIMP.
2. Создайте изображение размером 640×480 пикселей с разрешением 72 dpi. При этом используйте цветовую модель RGB.

3. Используя различные инструменты рисования, создайте изображение. Обязательно используйте кисти различных форм и размеров, различные режимы наложения цветов, специальные эффекты. Также используйте ластик, заливку и градиентную заливку.
4. Полученное изображение сохраните в различных форматах: xcf, bmp, tif (используя LZW компрессию), png, gif, gif с градациями серого цвета, jpg (с различной степенью сжатия: 90, 60, 40).
5. Создайте отчет в тестовом редакторе MS Word вставьте в отчет изображения из полученных файлов и запишите после каждого размер полученного файла.
6. Создайте изображение 20×20 пикселей.
7. Увеличьте масштаб отображения и создайте рисунок с эмблемой какой либо компании, используя ограниченное число цветов.
8. Сохраните полученное изображение в различных форматах: bmp, tif (используя LZW компрессию), gif, jpg (с различной степенью сжатия: 90, 60, 40).
9. Добавьте в отчет полученный изображения из сохранённых файлов. Запишите после каждого изображения размер полученного файла.
10. Проанализируйте полученные результаты в развернутом выводе. При этом оценивайте качество полученных изображений и размеры файлов.

2. ЛАБОРАТОРНАЯ РАБОТА «ФОТОМОНТАЖ»

Выделение областей

Выделение области является одним из важнейших этапов работы с изображением. С помощью выделения области выделяют объекты на изображении что бы использовать их в дальнейшем для фотомонтажа. Выделенные области можно заливать цветом, текстурой, градиентом. С выделенными областями можно проводить отдельную цветокоррекцию и к ним можно применять фильтры. Выделенная область обычно отображается в виде пунктирной рамки.

Неудивительно, что для выделения областей существует ряд приемов и инструментов (рис. 2.1).



Рис. 2.1. Инструменты выделения

Инструменты выделения предназначены для выделения областей активного слоя, чтобы можно было работать только с ними, не трогая всё остальное. Однако слои и работа с ними будут рассмотрены далее.

Прямоугольное и эллиптическое выделение

Инструменты прямоугольное и эллиптическое выделение позволяют выделять прямоугольные и эллиптические области соответственно. Это самые простые, но очень часто используемые типы выделения.

Для того что бы воспользоваться инструментом выделите его (рис. 2.1. Инструмент 1 или 2) на панели инструментов и далее выделите с помощью мыши область на изображении. После этого выделенная область будет отображаться в виде пунктирной рамки (рис. 2.2).

После этого размер выделенной области можно менять для этого используются области в углах выделения. Также выделенную область можно спокойно переносить, не боясь испортить изображение.

Более подробно параметры выделения можно задать в свойствах инструмента (рис. 2.3).

Как видим, что через параметры можно задать точную позицию размещения выделения и точные размеры. После задания размеров,

можно установить параметр «Фикс.» означающие фиксировать, напри-

мер, соотношение сторон. Это позволяет делать выделение, например, по размеру печати фотографий 10:15 и т. п.



Рис. 2.2. Выделенная область

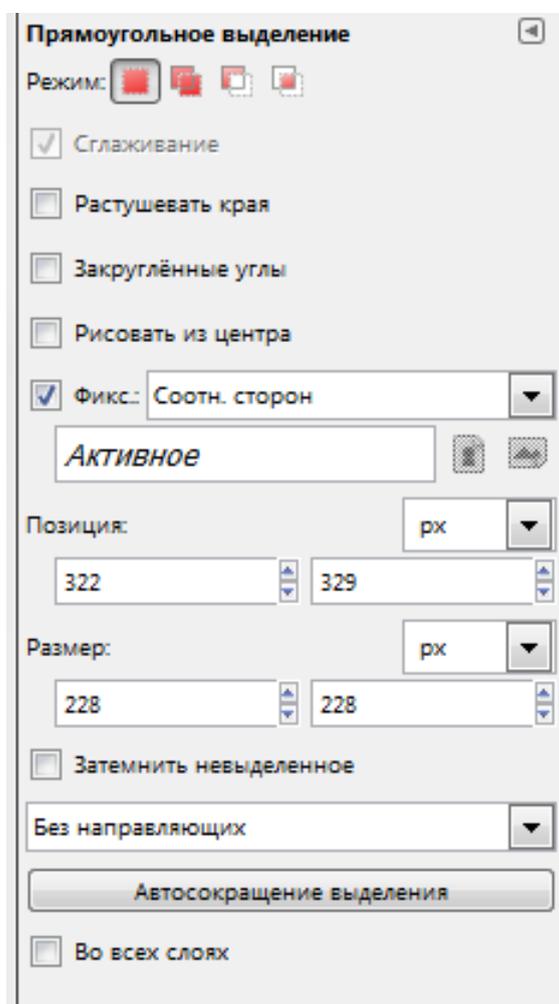


Рис. 2.3. Параметры инструмента «Прямоугольное выделение»

Наиболее важным параметром при сложных выделениях является **Режим** (рис. 2.4).

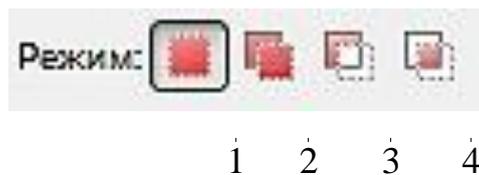


Рис. 2.4. Режимы выделения

Первый режим называется «Заменить текущее выделение» (рис. 2.4. кнопка 1). В таком режиме каждое выделение происходит заново, а предыдущее выделение снимается. Остальные режимы учитывают какое выделение было сделано до этого. Второй режим «Добавить в текущее выделение» к уже сделанному выделению добавляет новую область. Причем можно не выбирать данный режим в свойствах инструмента выделения а при выделении удерживать кнопку **Shift**. Третий режим позволяет вырезать из сделанной области какую либо часть. В этом режиме можно работать удерживая кнопку **Ctrl**. Последний режим позволяет найти пересечение.

Комбинируя режимы и инструменты можно создать выделенную область достаточно сложной формы, например, рис. 2.5.

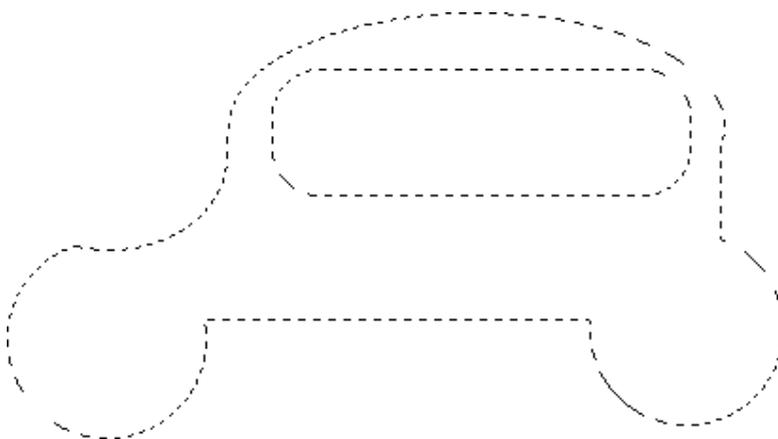


Рис. 2.5. Пример использования режимов выделения

На основе использования этих приемов создаются области выделения для разнообразных графических кнопок, размещаемых на Web-сайтах. Более подробно эту информацию можно найти в Internet набрав поисковый запрос «уроки создания кнопки в GIMP».

Свободное выделение и работа с быстрой маской

Инструмент **Свободное выделение** (рис. 2.1. Кнопка 3) позволяет выделять свободные области (рис. 2.6) при удержании левой кнопки мыши. При однократных нажатиях выделение происходит с помощью многоугольника.



Рис. 2.6. Пример выделения с помощью инструмента «Свободное выделение»

После такого выделения часто возникает необходимость коррекции области выделения, что удобно сделать перейдя в режим быстрой маски нажав соответствующую кнопку (рис. 1.3 кнопка 5) или сочетания **Shift+Q**. В этом режиме выделенная область остается цветной, а не выделенная отображается красным (рис. 2.7).



Рис. 2.7. Режим быстрой маски

В этом режиме для изменения границ выделенной области используются инструменты рисования: кисть для добавления к красной области и ластик для расширения области выделения.

Выход из режима быстрой маски осуществляется таким же образом как и вход.

Умные ножницы

Инструмент умные ножницы (рис. 2.1. кнопка 6) используется для выделения объектов по краю. Для этого необходимо расставить ряд опорных точек по краю объекта (рис. 2.8) и замкнуть линию и щелкнуть внутри области. При этом GIMP пытается самостоятельно определить цветовые границы. Этот метод хорошо работает, когда выделяемый объект не сливается с другими по цвету. Однако завершающим этапом выделения рекомендуется использовать доработку области выделения в режиме быстрой маски.



Рис. 2.8. Выделение с помощью умных ножниц

Выделение по цвету

Часто возникает необходимость выделения, какой либо области пикселей похожих по цвету. Это например необходимо для выделения объектов на однородном фоне. В этом случае выделяют фон, а потом инвертируют выделение (командой из главного меню **Выделение** → **Инвертировать**). Таким образом, оказывается выделенным сам объект.

Для выделения пикселей близких по цвету используется так называемая волшебная палочка (рис. 2.1. кнопка 4). Так один щелчок на синей области (рис. 2.9) может привести к выделению, показанному на рисунке 2.9.

При выборе данного инструмента самым главным параметром является **Порог**, который определяет чувствительность выделения к цветам. Так снижение порога до значения 4,0 в выше показанном примере может существенно сократить выделяемую область (рис. 2.10).



Рис. 2.9. Результат использования «Волшебной палочки»



Рис. 2.10. Результат снижения значения порога

И последнее: снятие выделения происходит при выборе **Выделение** → **Снять** или нажатии **Shift+Ctrl+A**.

Работа со слоями

Удобно представлять изображение в GIMP как пачку прозрачных листов: В терминологии GIMP, каждый прозрачный лист носит название слой. В принципе, нет ограничений на количество изображений в слое: единственное ограничение это количество доступной памяти в системе.

В GIMP границы слоя необязательно равны границам его содержащего изображения. Когда вы создаёте текст, к примеру, каждый текстовый элемент располагается в своём отдельном слое, и слой равен размеру текста, не больше. Также когда вы создаёте новый слой с помощью вырезания и вставки, новый слой создаётся достаточного размера для размещения вставленного содержимого. В окне изображения границы текущего активного слоя показаны чёрно-жёлтой пунктирной линией.

Структура слоёв в изображении показана в диалоге «Слой» (рис. 2.11).

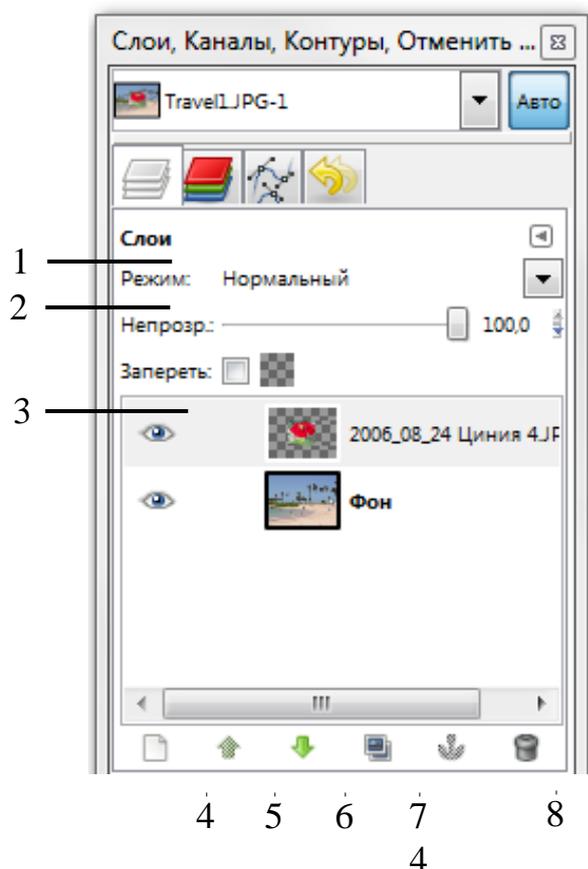


Рис. 2.11. Диалог «Слой»

В диалоге «Слой» можно изменять следующие свойства выделенного слоя:

Непрозрачность

Прозрачность слоя определяется степенью доступных цветов из нижних отображаемых слоёв списка. Непрозрачность определяется диапазоном от 0 до 100, где 0 означает полную прозрачность, и 100 означает полную непрозрачность. Непрозрачность определяется в диалоге Слой (рис. 2.11., элемент 2).

Видимость

Существует возможность временно не отображать слой без его уничтожения, с помощью щелчка по пиктограмме глаза (рис. 2.11, кнопка 3) в диалоге слоёв. Это называется «переключением видимости» слоя. Для большинства операций над изображением отключение видимости равносильно отсутствию слоя. Когда вы работаете с изображением, содержащим множество слоёв с разной прозрачностью, чаще вам будет проще получить лучший вид слоя, на котором вы в данный момент работаете отключением видимости других слоёв.

Режим

Режим слоя (рис. 2.11. элемент 1) определяется способом комбинации цветов из текущего и расположенного ниже слоя для представления видимого результата.

Режимы слоя иногда называются «режимами смешивания». Выбор режима слоя изменяет внешний вид слоя или изображения в зависимости от низлежащих слоёв. Если есть только один слой, то режим слоя ни на что не влияет. Поэтому должно быть по крайней мере два слоя, чтобы использовать режимы слоя.

Кнопки внизу диалога «Слои» позволяют создавать новые слои (рис. 2.11. кнопка 4), изменять порядок следования слоев (рис. 2.11. кнопки 5,6), создавать копию слоя (рис. 2.11 кнопка 7), удалять выделенный слой (рис. 2.11 кнопка 8).

Одним из этапов фотомонтажа обычно является создание нового слоя (рис. 2.11 кнопка 4) на изображении, а затем вставка выделенного объекта из другого изображения. Для вставки может использоваться буфер обмена и стандартные команды **Правка** → **Копировать**, **Правка** → **Вставить**. Более подробно фотомонтаж рассмотрим в следующих разделах.

Текст в GIMP

На изображение может быть добавлен любой текст с помощью инструмента «Текст» (рис. 2.12).



Рис. 2.12. Инструмент «Текст»

Добавление текста происходит в специальный текстовый слой. И сам текст может быть отредактирован в дальнейшем с помощью того

же инструмента «Текст». При выборе инструмента можно задать параметры шрифта (рис. 2.13) такие как: шрифт, размер, цвет.

После применения инструмента текст появляется специальный диалог для ввода и редактирования текста (рис. 2.14).

Размещение текста по контуру и создание контуров в данном пособии не рассматриваются.

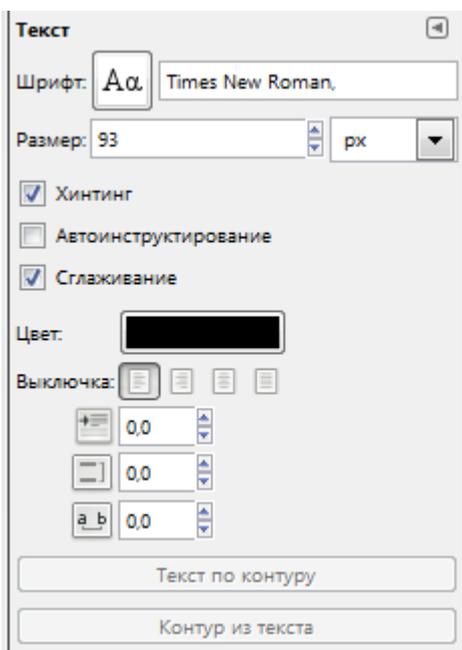


Рис. 2.13. Параметры инструмента «Текст»

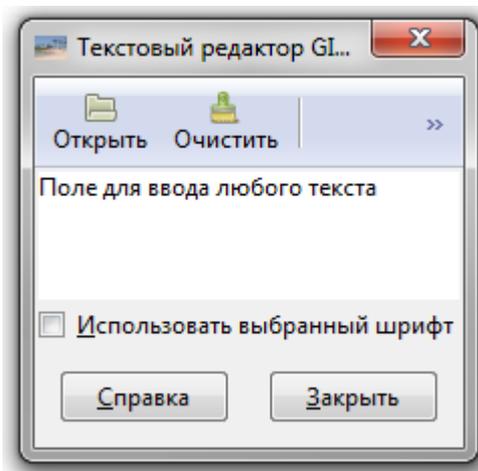


Рис. 2.14. Диалог для ввода и редактирования текста

Преобразование изображения в слой

Для преобразования слоя существует ряд инструментов (рис. 2.15).

Некоторые из приведенных инструментов можно применить как к отдельному выделенному слою, выделению или в целом к изображению.



1 2 3 4 5 6 7 8

Рис. 2.15. Инструменты преобразования

Общие свойства инструментов преобразования

Перед изучением инструментов преобразования заметим, что некоторые параметры для этих инструментов являются общими. Во первых это группа кнопок «Преобразование» (рис. 2.16).



1 2 3

Рис. 2.16. Группа кнопок преобразовать

При выборе первой кнопки инструмент работает над активным слоем. Если в слое есть выделение, то выделенная часть изображения будет трансформирована.

При выборе второй кнопки инструмент работает только над формой самого выделения, а не изображением в этом выделении.

При выборе третьей кнопки инструмент работает только над контуром.

Инструменты преобразования

Кратко рассмотрим назначение инструментов преобразования.

Инструмент «Перемещение» (рис. 2.15 кнопка 1) служит для переноса активного слоя, выделения или контура.

Инструмент «Кадрирование» (рис. 2.15 кнопка 3) служит для удаления областей с края изображения или слоя. Чаще всего этот инструмент применяется не к слою, а в целом ко всему изображению, перед выводом на печать для задания нужных соотношений сторон, например 10×15. Для этого в параметрах инструмента устанавливается галочка в положении «Фикс.», а в соответствующем списке должно быть установлено «Соотн. сторон». Для обрезки только текущего слоя в параметрах инструмента устанавливается галочка в опции «Только текущий слой».

Поскольку при фотомонтаже размеры изображений сильно отличаются необходимо провести коррекцию масштаба отдельных объектов находящихся в разных слоях. Для этого применяют инструмент «Масштаб» (рис. 2.15 кнопка 5). При выборе этого инструмента и щелчке на изображении появляется дополнительный диалог «Масштаб» (рис. 2.17).

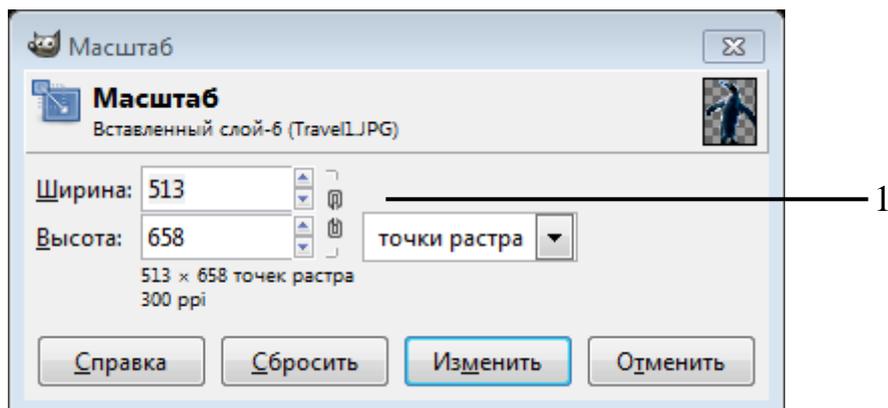


Рис. 2.17. Диалог «Масштаб»

Если возникает необходимость сохранений пропорций при масштабировании необходимо нажать на элемент в виде цепочки (рис. 2.17. область 1), что бы она обрела вид единого целого. Далее, применяя перетаскивание за углы выделенного объекта на слое, производится масштабирование. Окончательно преобразование подтверждается нажатием кнопки «Изменить» в диалоге масштаб. Подобные диалоги появляются и при других преобразованиях.

Инструмент «Вращение» (рис. 2.15 кнопка 4) служит для поворота активного слоя, выделения или контура. При активном инструменте поворот может осуществляться как с помощью мыши, так и через соответствующий диалог «Вращение».



Рис. 2.18. Изображения с двумя копиями объекта

Следующие инструменты «Искавление» (рис. 2.15 кнопка 6), служащий для схода изображения, «Перспектива» (рис. 2.15 кнопка 7), «Зеркало» (рис. 2.15 кнопка 8) выполняют соответствующие преобразования аналогичным образом. Они могут быть использованы для построения перспективной тени выделенного объекта следующим образом. Сначала слой дублируют с помощью кнопки на панели слоев (рис. 2.11. кнопка 7) и получают две копии объекта (рис. 2.18).

Далее используем инструменты «Перспектива», «Искавление» при необходимости «Зеркало» один слой трансформируем и получаем следующее:



Рис. 2.19. Пример использования инструментов преобразования



Рис. 2.20. Результат работы с копией слоя

Выделим объект в преобразованном слое. Для этого можно использовать волшебную палочку выделив сначала фон, а затем инвертировав выделение выбрав **Выделение** → **Инвертировать**. После этого зальем выделение черным цветом, используя инструмент «Плоская заливка» с установленным параметром «Все выделение» либо с нажатой кнопкой **Shift**. Далее остается переместить слой с помощью инструмента «Перемещение» и установить его прозрачность и получим:

Фотомонтаж

Все сведения для выполнения фотомонтажа уже изложены выше, приведем лишь общую схему выполнения действий. На первом этапе осуществляется подбор исходных изображений и открытие их в GIMP. Далее с помощью различных приемов и инструментов происходит выделение объектов и копирование их в буфер обмена при выборе команды **Правка** → **Копировать**, либо нажатии **Ctrl+C**. После этого переходим к фоновому изображению и выполняем вставку объекта (команда **Правка** → **Вставить** либо нажатие **Ctrl+V**). Объект в таком случае вставляется как плавающее выделение и лучше всего сразу создать для него новый слой нажав на панели «Слои» кнопку «Создать новый слой» (рис. 2.11. кнопка 4). После этого используются инструменты преобразования: «Масштаб», «Перемещение» и т. д., устанавливаются режимы слоя и прозрачность.

Описанные действия повторяются и для других объектов с других изображений. При необходимости меняется порядок следования слоев с панели «Слои» с помощью соответствующих кнопок (рис. 2.11. кнопки 5,6).

Для того что бы информация о слоях не была утеряна, полученное изображение рекомендуется сохранять в формате GIMP – xcf.

Задание по лабораторной работе

1. Подберите 3–4 исходных изображений. Добавьте найденные изображения в отчет по лабораторной работе.
2. Выполните фотомонтаж. Результат сохраните в отчет.
3. Добавьте тени от объектов как описано выше. Результат этого этапа добавьте в отчет.
4. Добавьте текстовые надписи. Полученное изображение вставьте в отчет.
5. Сохраните полученный файл в формате xcf.

3. ЛАБОРАТОРНАЯ РАБОТА «ОБРАБОТКА ИЗОБРАЖЕНИЙ»

Коррекция цвета

Для коррекции цвета в GIMP существует ряд инструментов в меню «Цвет» (рис. 3.1).

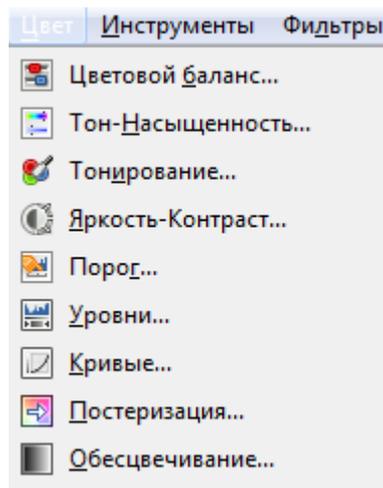


Рис. 3.1. Инструменты для коррекции цвета

Инструменты можно использовать для активного слоя или для участка выделения.

Цветовой баланс

Первый инструмент «Цветовой баланс» позволяет регулировать соотношение основных цветов из RGB и CMY моделей. Инструмент наиболее полезен для исправления цветов, преобладающих на цифровых фотографиях. Регулировка происходит либо для светлых участков изображения, либо для полутонов, либо для теней с помощью специального диалога (рис. 3.2).

Коррекция тона, освещенности, насыщенности

Следующий инструмент позволяет изменять значения тона, насыщенности и яркости выбранного цветового диапазона в активном слое или выделении с помощью специального диалога (рис. 3.3).

Тон позволяет отличать между собой основные цвета в модели HSV: красный, зеленый, синий, голубой, малиновый (пурпурный), желтый.

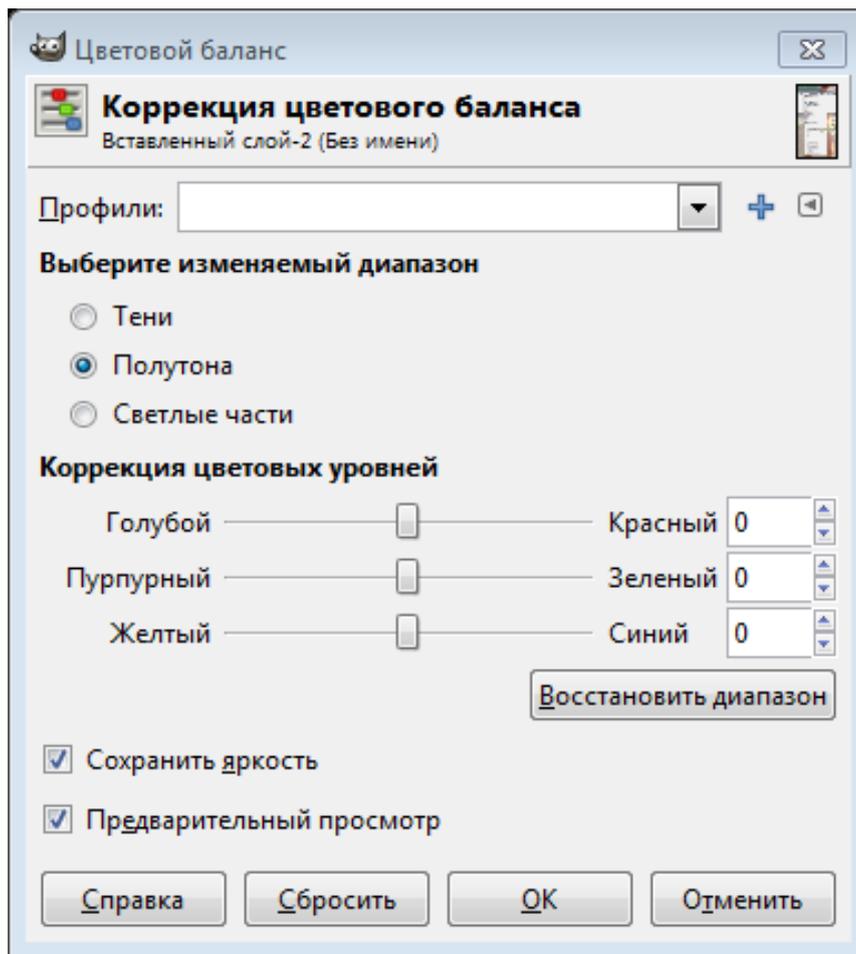


Рис. 3.2. Диалог для корректировки цветового баланса

В теории цвета **насыщенность** – это интенсивность определённого тона. Можно сказать, что насыщенность это характеристика цвета, определяющая его чистоту. Насыщенный цвет можно назвать сочным, глубоким, менее насыщенный – приглушённым, приближённым к серому. Также насыщенность позволяет отличать красный цвет от розового, зеленый от светло-зелёного и т. д.

Полностью ненасыщенный цвет будет оттенком серого. Насыщенность (saturation) – одна из трёх координат в цветовых пространствах моделей HSL и HSV.

Термин «освещенность» (от англ. lightness) в GIMP переведен на русский язык не совсем корректно. В русском языке в теории цвета используют обычно термин **светлота**. **Светлота** – одна из основных характеристик цвета наряду с насыщенностью и тоном. Светлота это субъективная яркость участка изображения, позволяющая отличать, например, серый цвет от черного, а белый от серого. Для полного понимания этих характеристик цвета рекомендуется обратиться к лекции по основам компьютерной графики и цветовым моделям HSV и HSL.

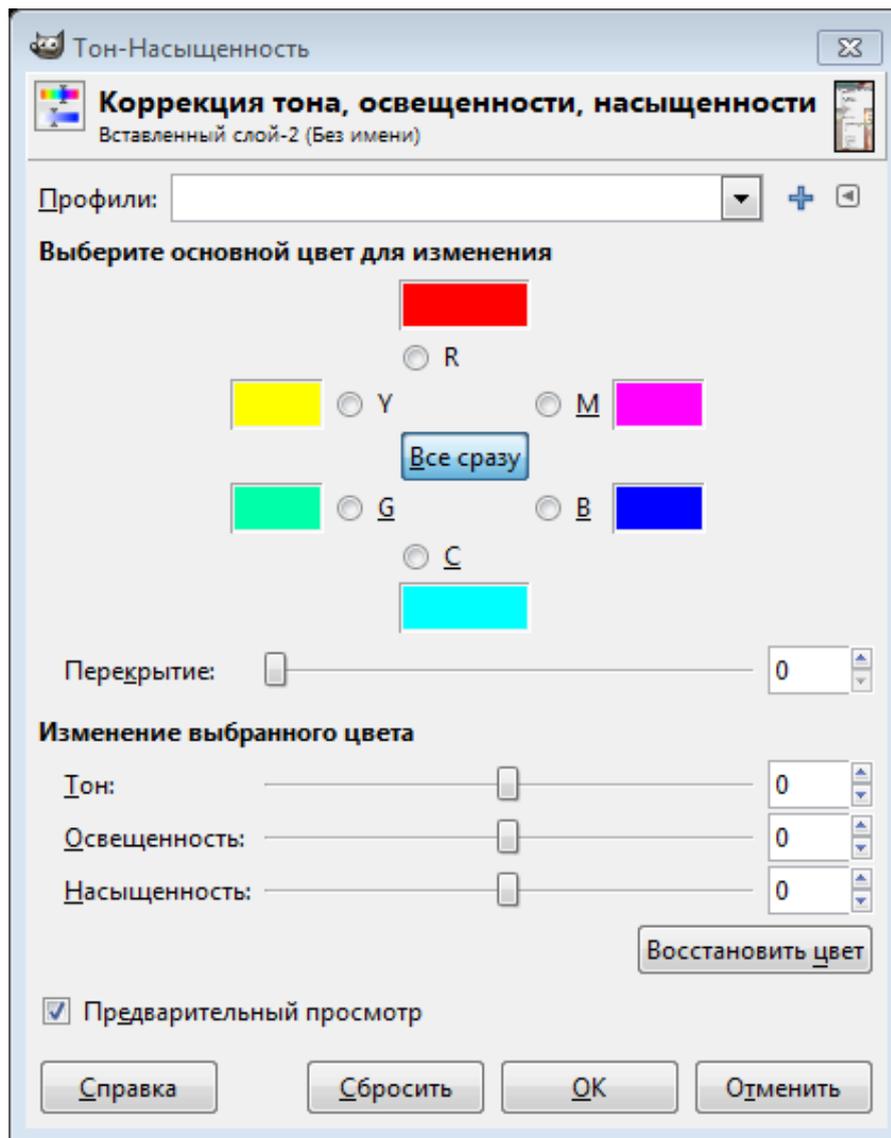


Рис. 3.3. Диалог коррекции тона, освещенности, насыщенности

Тонирование

Инструмент «Тонирование», так же можно использовать для изменения тона, освещенности и насыщенности через специальный диалог (рис. 3.4). Но в отличие от предыдущего инструмента, тонирование применяется для изображений в градациях серого цвета для получения цветности и может быть использовано для перевода изображения в сепию (дуотон). Использование инструментов выделения отдельных областей и тонирования позволяют черно-белую фотографию (изображение в градациях серого цвета) перевести в цвет. Также для раскрашивания черно белых изображений можно использовать команду **Цвет → Окрашивание**.

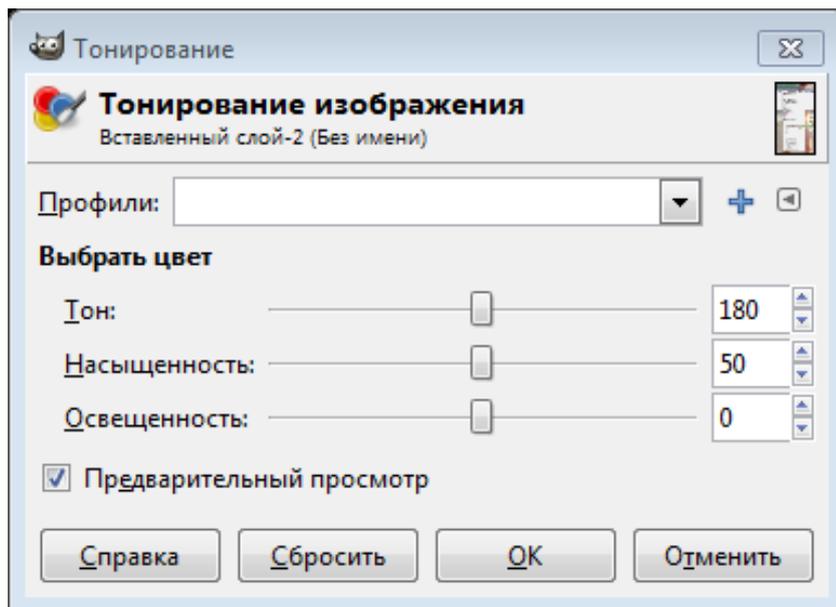


Рис. 3.4. Диалог «Тонирование»

Яркость и контраст

Достаточно просто изменяется яркость и контрастность изображения или выделения при использовании инструмента. «Яркость-контраст».

Яркость и **контраст** являются субъективными характеристиками изображения, воспринимаемыми человеком.

Яркость представляет собой характеристику, определяющую то, на сколько сильно цвета пикселей отличаются от чёрного цвета. Например, если оцифрованная фотография сделана в солнечную погоду, то ее яркость будет значительной. С другой стороны, если фотография сделана вечером или ночью, то её яркость будет невелика.

Контраст представляет собой характеристику того, насколько большой разброс имеют цвета пикселей изображения. Чем больший разброс имеют значения цветов пикселей, тем больший контраст имеет изображение.

Гистограмма изображения

Несколько следующих команд в меню «Цвет» используют **гистограмму** изображения (уровни) как основной или дополнительный инструмент. **Гистограмма** (в фотографии) – это график распределения полутонов изображения, в котором по горизонтальной оси представлена Яркость, а по вертикали – относительное число пикселей с данным значением яркости (рис. 3.5). Гистограмма изображения позволяет оценить количество и разнообразие оттенков изображения, а также общий уровень яркости изображения.

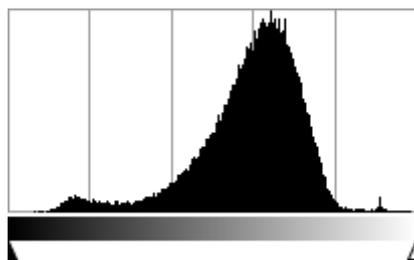


Рис. 3.5. Пример гистограммы

Наиболее просто построение гистограммы можно пояснить для изображения в градациях серого цвета. В этом случае, гистограмма представляет собой диаграмму, где по горизонтальной шкале откладываются градации серого от 0 (черный) до 255 (белый), а по вертикальной – количество точек соответствующей градации в этом изображении. Чем выше столбец, тем больше точек соответствующего оттенка серого содержится в фотографии.

Так гистограмма, приведенная на рис. 3.5., отображает количество пикселей определённого тона для рис. 3.6.

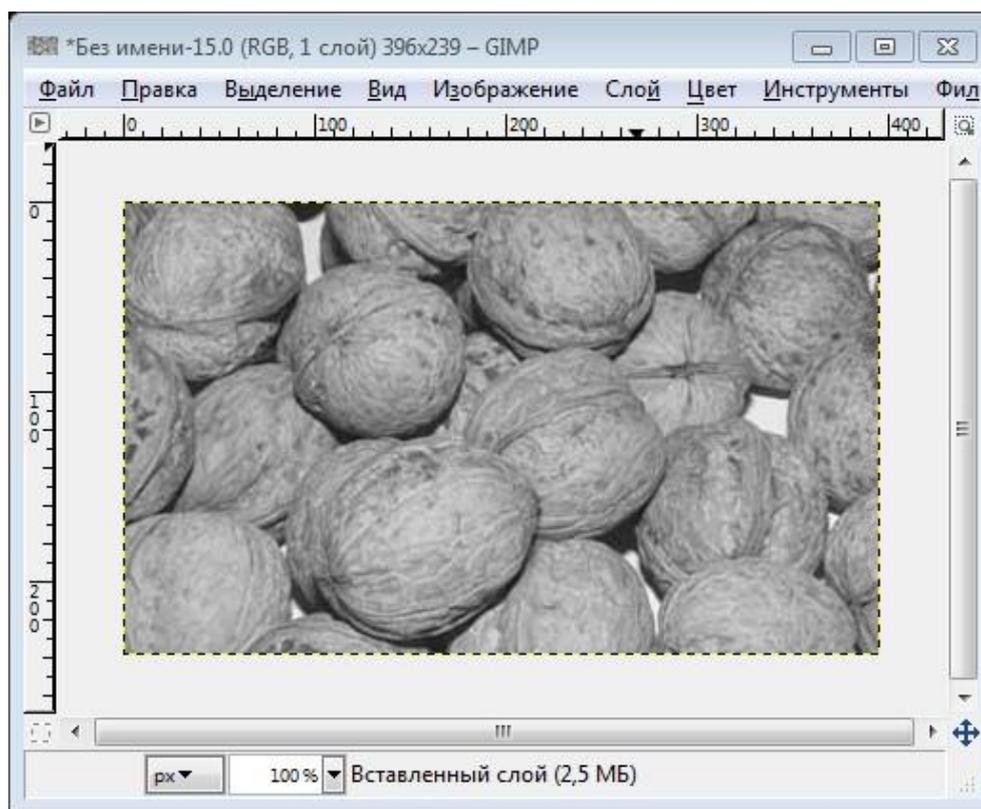


Рис. 3.6. Пример изображения для построения гистограммы

При выборе инструмента «Уровни» и задании параметров, как показано на рис. 3.7. (бегунки справа слева сдвинуты к середине), распределяем значение интенсивностей равномерно по всей области градаций серого цвета. Этим самым повышаем контрастность изображения (рис. 3.8).

В некоторых случаях, описанный метод позволяет улучшить изображение, а в некоторых наоборот уменьшить художественную ценность изображения.

Гистограмма для цветного изображения строится по яркости, либо по каждому отдельному каналу для основных цветов цветовой модели. Например, для RGB модели гистограмма может быть построена для трех каналов R, G и B соответственно.

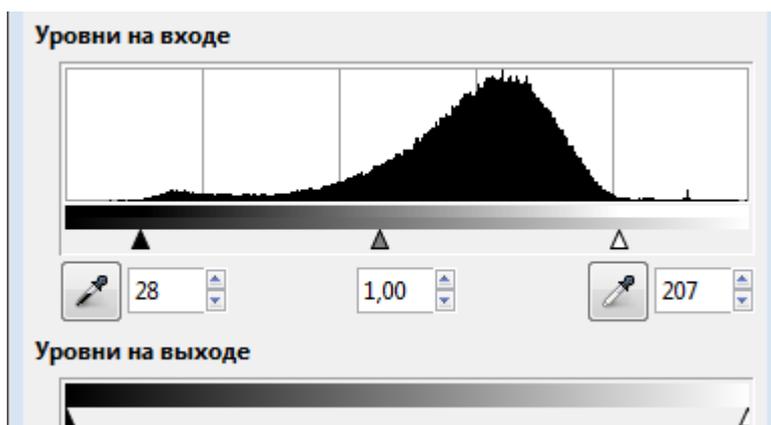


Рис. 3.7. Изменение гистограммы

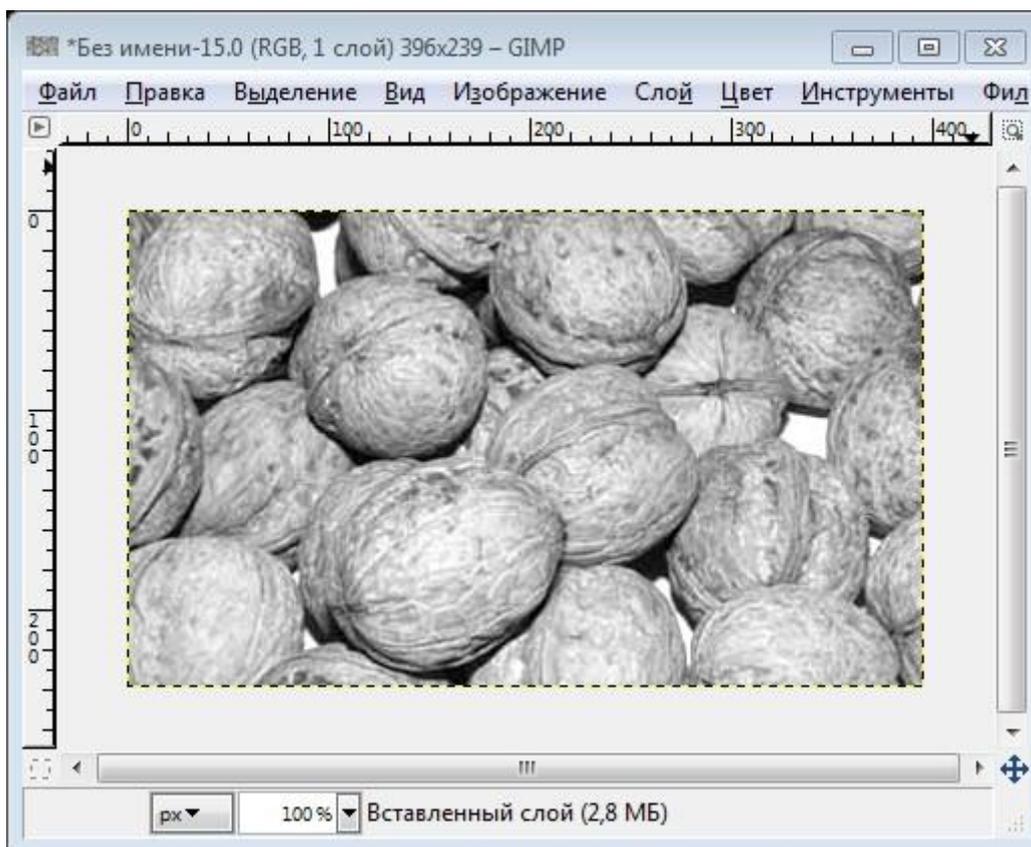


Рис. 3.8. Изображение с повышенной контрастностью

Коррекция цветových кривых

Вторым значимым элементом после гистограммы являются «Кривые», которые позволяют управлять функцией яркости и контрастности.

Рассмотрим функцию для управления яркостью и контрастностью, областью определения и значений которой являются значения цветových компонент в модели RGB. Аргументом функции является цвет пикселя исходного изображения. Значение функции представляет собой цвет пикселя обработанного изображения. Для изменения яркости/контраста функция применяется для каждого пикселя изображения.

Если яркость и контраст изображения никак не меняются в процессе преобразования, то функция имеет график, представленный на рис. 3.9., а. Из рисунка видно, что функция в этом случае просто передаёт на выход значение своего аргумента.

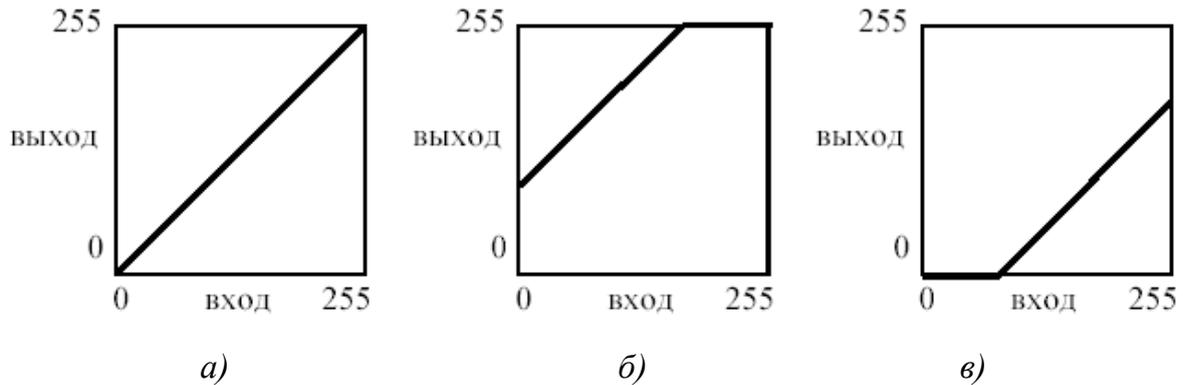


Рис. 3.9. Графики яркости

Яркость для рассматриваемой функции представляет собой сдвиг прямой линии в вертикальном направлении. Яркость изображения увеличивается пропорционально сдвигу прямой. Если прямая сдвигается вверх (рис. 3.9., б), яркость изображения увеличивается, а если прямая сдвигается вниз (рис. 3.9., в) – уменьшается.

При использовании преобразования контраста прямая линия меняет свой наклон. При увеличении контраста изображения (рис. 3.10., а) наклон прямой увеличивается, при уменьшении контраста – уменьшается (рис. 3.10., б). При этом сдвиг прямой в горизонтальном направлении означает, что помимо контраста изменяется и яркость изображения.

Комбинации наклона и сдвига прямой позволяют одновременно изменять и яркость, и контраст изображения. Например, на рис. 3.11. представлен график функции, усиливающей контраст и увеличивающей яркость изображения.

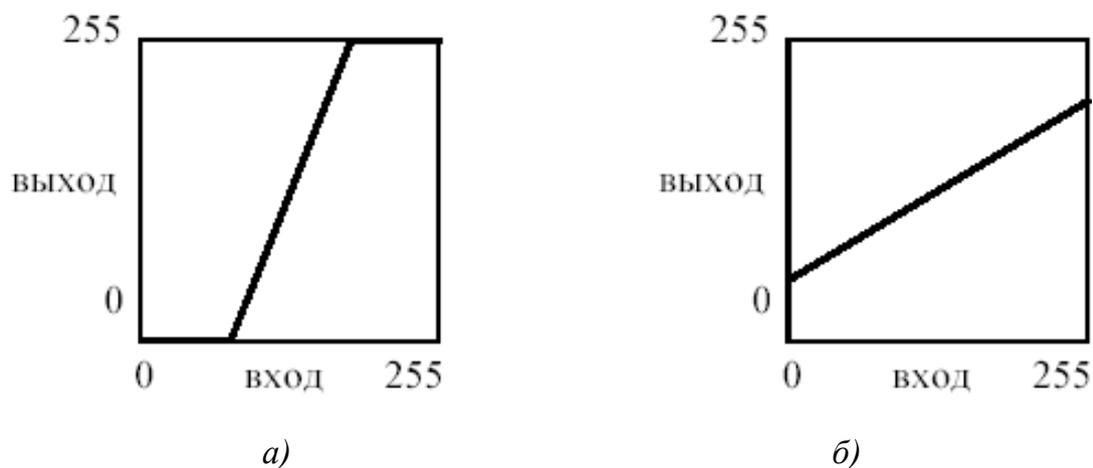


Рис. 3.10. Графики контрастности

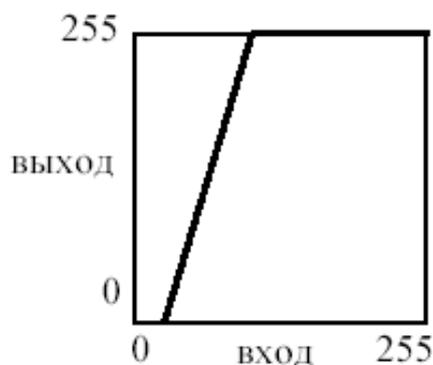


Рис. 3.11. Увеличение яркости и контрастности

Преобразование яркости/контраста может быть применено и к отдельным компонентам модели RGB, например к компоненту красного цвета. Тогда яркость/контраст будут изменяться только для красного компонента, а для других компонент они останутся неизменными. Более того, можно задавать различные преобразования яркости/контраста одновременно для каждого компонента модели RGB.

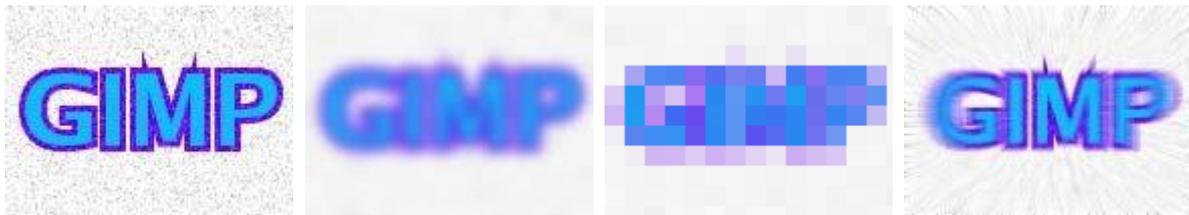
Фильтры

Фильтр – специальный вид инструмента, который берёт входной слой или изображение, применяет к нему математический алгоритм и возвращает измененный слой или изображение в новом формате. Фильтры позволяют накладывать на изображение различные эффекты, например: размытие, резкость, деформацию, шум и т. д.

Для работы с фильтрами в GIMP выделено специальное меню «**Фильтры**». При работе с фильтрами активно используются диалоговые окна для задания параметров фильтров.

Фильтры размытия

Это набор фильтров, которые тем или иным способом размывают изображение или его часть. Тем не менее, цвета необработанной области могут попасть в размытую область. Так что ниже приведены иллюстрации действия каждого из фильтров размывания, которые помогут вам выбрать фильтр, оптимально подходящий для той или иной задачи. Разумеется, это всего лишь примеры, поскольку почти для каждого фильтра можно изменить тип размывания и силу действия эффекта.



а) Исходное изображение

б) Размывание по Гауссу

в) Пикселизация

г) Размывание в движении

Рис. 3.12. Примеры использования фильтров размытия

На рис. 3.12 приведены примеры использования различных фильтров размытия.

Фильтры улучшения

Среди фильтров улучшения можно выделить фильтр повышения резкости, удаления пятен и штрихов, и самое главное, удаления эффекта красных глаз на цифровых фотографиях. Для использования последнего фильтра рекомендуется сначала выделить область, с эффектом красных глаз на фотографии, и далее применить фильтр, меняя пороговое значение в диалоговом окне.

Фильтры искажения

Фильтры искажения преобразуют изображение разными способами, такими как: имитация ветра, ряби или волн на воде, загнутая страница, искажения оптики и т. д.

Фильтры свет и тень

Здесь находится три группы фильтров:

- Фильтры световых эффектов рисуют разные эффекты освещения изображения.
- Фильтры для создания разного рода теней. Необходимо отметить, что описанный выше способ получения тени через работу со слоя-

ми, более гибко и позволяет получать более сложные тени, например с изгибом на полу и стене.

- Фильтры эффекта стекла искажают изображение так, как будто на него смотрят сквозь линзу или стеклянные блоки.

Фильтры выделения края

Фильтры выделения края ищут границы между разными цветами, таким образом, находя контуры объектов.

Они используются, чтобы указать выделения и для других художественных целей. Например, интересен фильтр «Неон».

Фильтры имитации

Фильтры имитации создают эффекты присущие различным стилям живописи: кубизму, живописи маслом, картине на холсте или плетённой поверхности и т. д.

Фильтры визуализации

Большинство фильтров в GIMP работает над слоем, изменяя его содержимое, но фильтры в группе «Визуализация» отличаются тем, что они создают текстуры с нуля. Обычный результат такого фильтра – полная замена содержимого слоя. Некоторые фильтры создают случайные или шумовые текстуры, другие – фракталы, а один (Gfig) больше напоминает общий (но ограниченный) инструмент векторной графики.

В этой же группе фильтров находятся фильтры для построения и изучения фракталов. При выборе **Фильтры → Визуализация → Природа → IFS-фрактал** вызывается подсистема построения геометрических фракталов с помощью системы итерируемых функций. При выборе фильтра **«Исследователь фракталов»** вызывается подсистема построения разнообразных алгебраических фракталов. Для этих фильтров GIMP содержит достаточно подробную справку с пошаговыми инструкциями.

Задание по лабораторной работе

1. Подобрать исходное изображение в градациях серого цвета. Поместить изображения в отчет.
2. Перевести изображения в режим RGB модели, выбрав **Изображение → Режим → RGB**.
3. Используя инструменты выделения, тонирование и окрашивание сделать изображение цветным.
4. Отрегулировать цветовой баланс, яркость и контрастность.
5. Сохранить полученное изображение. Описать ход работы и полученные результаты.

6. Подобрать несколько разных цветных изображений и исследовать изображения с помощью гистограммы.
7. При необходимости увеличить контрастность изображений. Отразить результаты исследования в отчете, в котором привести исходные и полученные изображения и их гистограммы.
8. Создать новое изображение и залить его градиентом от черного к белому цвету, слева направо.
9. Используя коррекцию цветových кривых как показано на рис. 3.13. Преобразовать изображение. Объяснить результат в отчете.

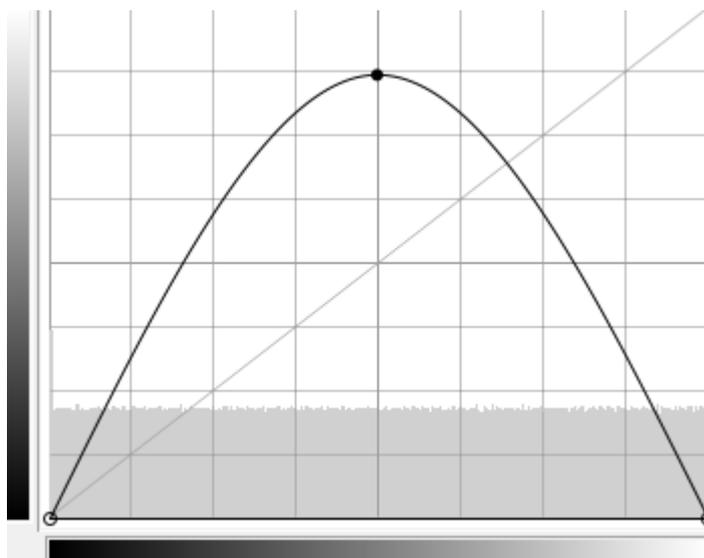


Рис. 3.13. Кривая цветокоррекции

10. Подобрать фотографию с эффектом «Красных глаз».
11. Устранить эффект «красных глаз» с помощью соответствующего фильтра.
12. Кадрировать изображение, подготовив его к печати на фотобумаге размером 10×15 см.
13. Улучшить изображение, используя гистограмму (уровни).
14. Создать выделение по краям изображения. Для этого используйте прямоугольное выделение с закругленными краями. Для симметричности выделения в параметрах инструмента «Прямоугольное выделение» позицию и размер. Затем инвертируйте выделение командой **Выделение** → **Инвертировать (Ctrl + I)**.
15. Примените к выделению несколько различных фильтров для получения оригинальных рамок. Можно попробовать выбрать **Фильтры** → **Карта** → **Фрактальный след**. Хорошие результаты сохраните в отчет. Используйте историю для отмены неудачных действий и новых попыток.

16. Подберите цветные изображения для дальнейших экспериментов.
17. Исследуйте группы фильтров: **«Искажение»**, **«Выделение края»**, **«Имитация»** применяя их в целом ко всему изображению. Наиболее интересные результаты занесите в отчет.
18. Создайте новое изображение 640×480.
19. Исследуйте на этом изображении возможности группы фильтров **«Визуализация»**.
20. Исследуйте возможности построения геометрических и алгебраических фракталов.
21. Исследуйте возможности построения векторных примитивов на изображении с помощью фильтра Gfig.
22. Напишите развернутый вывод, где проанализируйте основные возможности GIMP в сравнении с другими графическими редакторами. Выразите и обоснуйте свое мнение.

4. ЛАБОРАТОРНАЯ РАБОТА «ОСНОВЫ INKSCAPE»

Inkscape (Инкскейп) – свободно распространяемый векторный графический редактор, поддерживающий открытый формат SVG.

Inkscape использует формат SVG (Scalable Vector Graphics – Масштабируемая векторная графика) для своих файлов. SVG является открытым стандартом и широко используется в графических пакетах. Формат SVG использует язык разметки XML, поэтому файлы в этом формате могут редактироваться любым текстовым или XML-редактором (отдельно от Inkscape). Помимо SVG, в Inkscape можно работать и с другими форматами (например, EPS и PNG).

Основы Inkscape

Окно Inkscape

На рис. 4.1. представлено основное окно редактора Inkscape.

Сверху расположено главное меню, содержащее основные команды. Слева находится панель инструментов с командами добавления и редактирования объектов. Под главным меню располагается панель свойств для выбранного инструмента. Справа располагаются панели содержащие дополнительные команды, в том числе команды для работы с документом. Внизу окна расположена палитра цветов и строка состояния.

Многие действия доступны с клавиатуры. Полный справочник по клавишам находится в меню Справка Использование клавиатуры и мыши».

Перемещение по холсту и изменение масштаба

Основной документ, с которым работает Inkscape, называется холст. Есть множество способов перемещаться по холсту. Для перемещения при помощи клавиатуры используйте Ctrl+стрелки. Также можно передвигаться по холсту, зажав его поверхность средней клавишей мыши, или при помощи полос прокрутки.

Изменять масштаб можно также с клавиатуры кнопками – или +. Кроме этого существует специальный инструмент для масштабирования



, которым можно увеличивать только необходимую выделенную область.

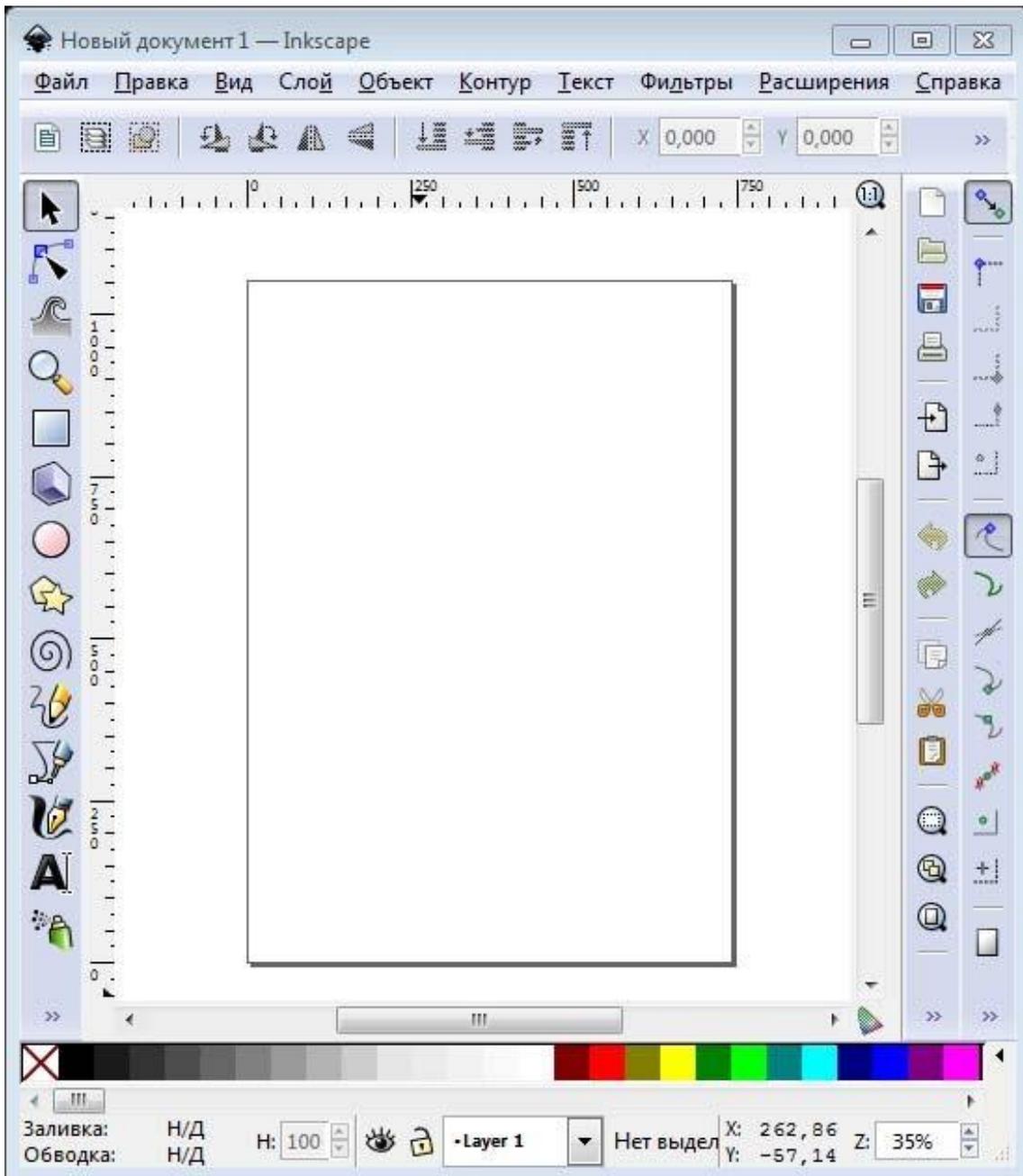


Рис. 4.1. Окно Inkscape

Инструменты Inkscape

Панель со значками в левой части окна представляет инструменты Inkscape для рисования и редактирования. В верхней части окна (под меню) находится Панель управления с основными командными кнопками и панель Параметры инструментов (чуть ниже панели управления), содержащая параметры, специфичные для каждого инструмента. Строка состояния (внизу окна) будет показывать полезные подсказки во время вашей работы.

Многие действия доступны с клавиатуры. Полный справочник по клавишам находится в меню **Справка** → **Использование клавиатуры и мыши**.

Работа с документами

Для создания нового документа используйте **Файл** → **Создать** или нажмите **Ctrl+N**. Чтобы открыть существующий документ SVG, используйте **Файл** → **Открыть** (**Ctrl+O**). Для сохранения используйте **Файл** → **Сохранить** (**Ctrl+S**), либо **Сохранить как...** (**Shift+Ctrl+S**) для сохранения файла под другим именем.

Фигуры

У Inkscape есть четыре удобных инструмента для работы с *фигурами*, каждый из которых может создавать или редактировать только собственный тип фигур. *Фигура* – это объект, изменять который можно разными уникальными для него способами, *узлы управления* и числовые *параметры*, которые определяют внешний вид фигуры.

Если взять, к примеру, звезду то можно менять количество лучей, её длину, угол, округлость и т. п. При этом звезда остаётся звездой. Фигура менее свободна» чем просто контур, но зачастую более интересна и полезна. Вы всегда можете преобразовать фигуру в контур (**Ctrl+Shift+C**), но обратное преобразование невозможно.

К инструментам фигур относятся инструменты для рисования прямоугольников , эллипсов , звёзд  и спиралей . Для начала давайте посмотрим на общие принципы работы с ними.

Основные приемы

Новая фигура создается нажатием и **перетаскиванием** по холсту курсора соответствующего инструмента. Когда фигура создана (и выбрана), она отображает свои узлы управления в виде белых меток в форме кристалла. Если фигура выбрана, то ее можно редактировать, перемещая эти узлы.

Все четыре разновидности фигур показывают свои узлы управления с любым из включенных инструментов редактирования фигур, также как и с включенным инструментом редактирования узлов (**F2**). В момент наведения курсора мыши на один из узлов он [курсор] сообщает вам в строке состояния, что сделает этот узел при его смещении или нажатии по нему с разными модификаторами.

Кроме того, каждая фигура отображает свои параметры в панели, которая находится над холстом. Обычно панель настроек инструмента

содержит несколько числовых полей для ввода и кнопку сброса значений в изначальное состояние.

Любые изменения в параметрах инструмента запоминаются и используются для следующей новой фигуры. Например, если изменить количество лучей звезды, у всех последующих новых звезд будет такое же количество лучей. Более того, новые параметры фигур запоминаются глобально для каждой новой сессии работы с Inkscape.

С включенным инструментом редактирования фигуры объект можно выбрать при помощи **щелчка (мыши)**. Комбинации **Ctrl+щелчок** (выбрать одну фигуру из группы) и **Alt+щелчок** (выбрать под фигурой) работают так же, как и в инструменте выделения. **Esc** сбрасывает все выделения.

Перемещение, изменение размера и вращение

Наиболее популярный инструмент в Inkscape – Селектор . Выбрать его можно щелчком по чёрной стрелке (либо нажав F1 или пробел). Этим инструментом вы можете выбрать любой объект на холсте. Щёлкните мышью по любому объекту (например квадрату). Вокруг объекта вы увидите восемь стрелок. Теперь вы можете:

- Передвигать объект (с нажатым Ctrl перемещения ограничиваются двумя осями: горизонтальной и вертикальной).
- Менять размер объекта, потянув за любую из стрелок (меняя размер с нажатым Ctrl, вы сохраните пропорции оригинала).

Щёлкните мышью по прямоугольнику ещё раз – направление стрелок изменится. Теперь вы можете:

- Поворачивать объект, потянув за угловые стрелки (с нажатым Ctrl объект будет поворачиваться шагами по 15 градусов; сместив крестик, вы сместите центр вращения).
- Перекашивать (наклонять) объект, двигая неугловые стрелки (с нажатым Ctrl перекашивание будет производиться с шагом в 15 градусов).

В этом режиме (режиме выделения объектов) вы так же можете менять размеры и расположение выделения на холсте, используя поля сверху.

Изменение формы при помощи клавиш

Одна из особенностей Inkscape, отличающая его от большинства других редакторов векторной графики – удобное управление с клавиатуры. Трудно найти команду или действие, которые было бы невозможно выполнить с клавиатуры, и изменение формы объектов – не исключение.

Вы можете использовать клавиатуру для перемещения объектов (клавиши-стрелки), изменения размера (клавиши < и >) и вращения (клавиши [и]). По умолчанию шаг перемещения и смены размера равен двум пикселям. С нажатой клавишей **Shift** это значение увеличивается в 10 раз (и становится равным 20 пикселям). Клавиши **Ctrl+>** и **Ctrl+<** увеличивают или уменьшают объект на 200 % или 50 % от оригинала соответственно. С нажатой клавишей **Ctrl** вращение будет выполняться с шагом в 90 градусов вместо 15.

Кстати говоря, наиболее удобны пиксельные изменения формы, производимые с нажатой клавишей **Alt** и клавишами изменения форм. Например, **Alt+стрелки** будут двигать выбранное на 1 пиксел *данного масштаба* (т. е. на 1 пиксел экрана, не путайте с пикселом, который является SVG единицей длины и отличается от пиксела масштаба). Это означает, что если вы увеличили масштаб, то **Alt+стрелка** даст *меньшее* смещение от абсолютного измерения, что по-прежнему будет выглядеть как смещение на пиксел на экране. Это даёт возможность точно разместить объект, изменяя масштаб.

Схожим образом **Alt+>** и **Alt+<** изменяют размер на один пиксел, а **Alt+[** и **Alt+]** вращают объект на один пиксел.

Выделение нескольких объектов

Вы можете выбрать любое количество объектов одновременно, нажав **Shift+щелчок** на желаемых объектах. Также можно выбрать объекты рамкой выделения – так называемым резиновым выделением (рамка выделения появляется тогда, когда выделение начинается с пустого места, а с нажатой клавишей **Shift** рамка выделения появится и над объектом).

Каждый выделенный объект отображается с пунктирной рамкой вокруг него. Благодаря этой рамке просто определить, какой объект выделен, а какой нет. Например, если выбрать оба эллипса и прямоугольник под ними, то без пунктирной рамки будет сложно понять, выделены эллипсы или нет.

Shift+щелчок на выделенном объекте исключает его из общего выделения. Попробуйте для практики выбрать три объекта сверху, а после этого, используя **Shift+щелчок**, исключите эллипсы, оставив выделенным только прямоугольник.

Нажатие **Esc** сбросит все выделения. **Ctrl+A** выделяет все объекты в пределах активного слоя (если вы не создавали слоёв, то это равносильно выделению всех объектов документа).

Группировка

Несколько объектов могут быть объединены в группу. При перемещении и трансформации группа ведёт себя также как и обычный объект.

Для создания группы нужно выбрать один или более объектов и выбрать **Объект** → **Сгруппировать (Ctrl+G)**. Разгруппировать их можно, нажав **Ctrl+U** или выбрав в меню **Объект** → **Разгруппировать** и предварительно выбрав группу. Сами по себе группы могут быть сгруппированы и как одиночные объекты. Подобная поэтапная группировка может быть сколько угодно сложной. При этом следует помнить, что **Ctrl+U** разгруппирует только последнюю группировку. Нужно нажать **Ctrl+U** несколько раз, если вы хотите полностью разгруппировать сложносгруппированные группы в группе.

Очень удобно то, что не нужно разбивать группу для редактирования отдельных объектов. Выполнив **Ctrl+щелчок** по объекту, вы его выберете и сможете редактировать. Таким же образом работает комбинация **Shift+Ctrl+щелчок**, позволяющая редактировать несколько объектов независимо от группы.

Заливка и обводка

Множество функций Inkscape доступны через диалоги (субменю). Вероятно, самый простой способ заполнить объект каким-либо цветом – это выбрать палитру внизу окна.

Но более грамотным способом будет выбор диалога «Заливка и обводка...» (рис. 4.2) через меню **Объект** → **Заливка и обводка (Shift+Ctrl+F)**.

Диалог содержит три вкладки: «Заливка», «Обводка», и «Стиль обводки». Вкладка «Заливка» позволит вам изменить заполнение выбранного объекта (или объектов). Используя кнопки под вкладкой, вы можете выбрать тип заливки, включая режим «Нет заливки» (кнопка со знаком X), режим «Сплошной цвет», режимы «Линейный градиент» или «Радиальный градиент».

Чуть ниже расположены кнопки-варианты выбора цвета. Каждый вариант имеет свою вкладку: RGB, CMYK, HSL, и «Круг». Вероятно, самым удобным вариантом является «Круг», в нём можно выбрать тон цвета, вращая треугольник, а затем подобрать насыщенность и яркость в самом треугольнике. Все варианты выбора цвета имеют возможность менять альфа-канал (прозрачность) выбранного объекта (или объектов).

Каждый раз при выборе объекта вкладка «Заливка и обводка...» показывает текущее значение для данного объекта (для нескольких одновременно выбранных объектов, вкладка цвета показывает их *усреднённый* цвет).

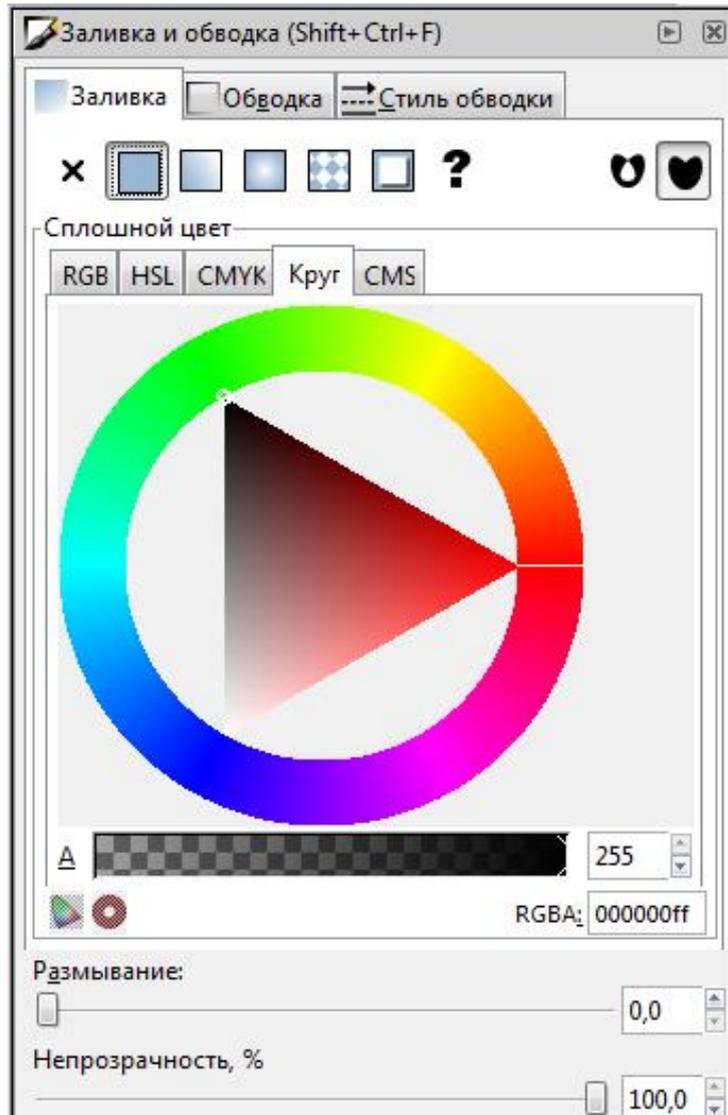


Рис. 4.2. Диалог «Заливка и обводка»

Используя вкладку «Обводка», вы можете задать границу объекта, установить ее цвет или прозрачность:

Последняя вкладка «Стиль обводки» позволяет изменить толщину и другие параметры границы.

Ещё один способ изменить цвет объекта – использовать инструмент Пипетка  (F7). Выбрав этот инструмент, щёлкните мышью в любой части рисунка, и полученный цвет будет присвоен выбранному до этого объекту (**Shift+щелчок** присвоит цвет границе объекта).

Дублирование, выравнивание, распределение

Одним из наиболее распространённых действий является дублирование объекта (**Ctrl+D**). Дублирование размещает дубликат над ориги-

налом и делает его выделенным так, что вы можете переместить его в сторону при помощи мыши или клавиш со стрелками.

Выделенные несколько объектов можно выравнивать и для этого существует специальный диалог «Выровнять и расставить» (рис. 4.3), вызываемый из меню **Объект** → **Выровнять и расставить** (**Ctrl+Shift+A**).

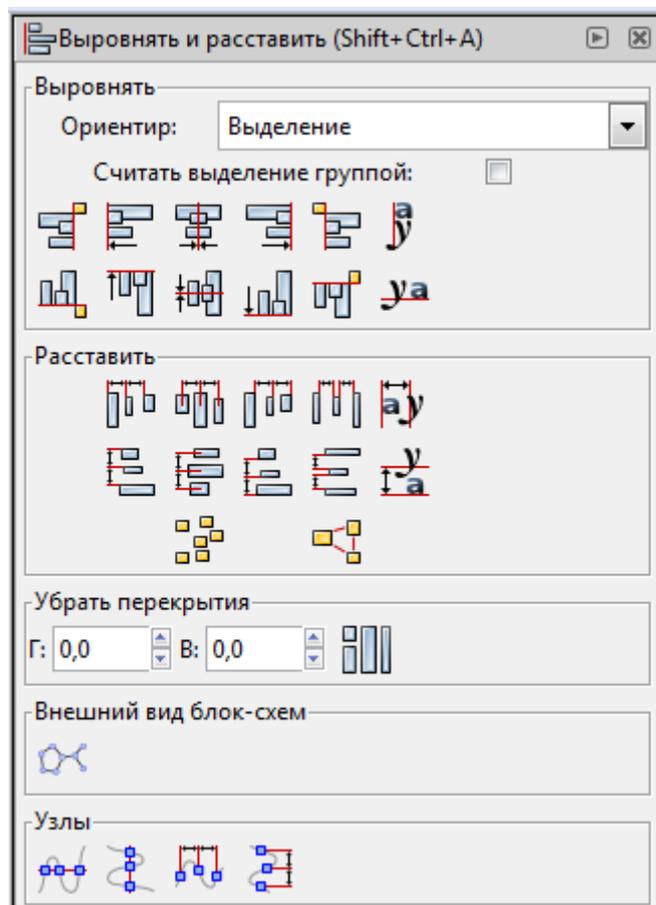


Рис. 4.3. Диалог «Выровнять расставить»

Z-порядок

Термин Z-порядок (порядок по оси Z) относится к перекрыванию объектами друг друга на рисунке. Иначе говоря, Z-порядок определяет, какой объект находится выше и закрывает собой другие. Две команды в меню **Объект** → **Поднять на передний план** (кнопка **Home**) и **Объект** → **Опустить на задний план** (кнопка **End**), переместят выбранный объект в самую верхнюю или самую нижнюю позицию по оси Z данного слоя. Две другие команды: **Поднять** (**PgUp**) и **Опустить** (**PgDn**) опустят или приподнимут выбранный объект (или объекты), *но только на один уровень* относительно других невыделенных объектов по оси Z (считаются только объекты, перекрывающие выделенные; ес-

ли выделение ничем не перекрывается, действие «Поднять» и «Опустить» будет ставить его в самую верхнюю или самую нижнюю позицию соответственно).

Задание по лабораторной работе

1. Запустите Inkscape.
2. Создать новый лист размером «По умолчанию».
3. Создайте следующие фигуры: Прямоугольник, эллипс, звезду, спираль, кривую Безье и текст.
4. Задайте различную заливку (в том числе и градиенты) и обводку (в том числе и пунктиры) созданным фигурам.
5. Экспортируйте изображение в растр и добавьте его в отчет.
6. Сохраните документ в формате SVG.
7. Откройте сохраненный *.SVG в текстовом редакторе (notepad).
8. Перенесите в отчет XML описания каждой фигуры. Опишите ключевые параметры каждой из фигур, влияющие на их положение, форму и цвет.
9. Проанализируйте сделанную работу в выводе. Также, опишите в выводе преимущества векторного формата изображения перед растровым и преимущества формата SVG.

5. ЛАБОРАТОРНАЯ РАБОТА «СОЗДАНИЕ ВЕКТОРНОГО ЛОГОТИПА»

Размещение текста вдоль контура

В Inkscape существует возможность размещения текста вдоль каких либо линий, в том числе и кривых и вдоль любой сложной фигуры, созданной путем преобразования фигуры в кривую. Сложная фигура, созданная путем слияния, автоматически становится кривой. Располагать текст вдоль фигур (прямоугольников, эллипсов и т. д.) нельзя, поэтому предварительно такие фигуры нужно обязательно преобразовывать в кривые. Например, это можно выполнить командой **Контур** → **Оконтурить объект**. Алгоритм размещения текста вдоль контура выглядит следующим образом:

1. Нажмите на иконку  и создайте текстовый объект.
2. Нажмите на кнопку для создания нужной фигуры. Нарисуйте фигуру.
3. Нажмите на кнопку  и выберите оба объекта. Выбрать оба объекта можно последовательно щелкнув по ним с нажатой клавишей **Shift**.
4. В меню, выберите **Текст** → **Разместить по контуру**

Выполнение логических операций над фигурами

Над несколькими выделенными объектами возможно выполнение логических операций сложения, вычитания, пересечения исключающее ИЛИ и т. п.

Сумма

Эта команда сливает два объекта и делает из них один. Может применяться к любому количеству объектов. Получаемый в результате выполнения операции объект всегда использует настройки стиля (заливки и штриха) нижнего объекта. Алгоритм сложения двух объектов приведен ниже.

1. Создайте две фигуры с помощью инструментов.
2. Нажмите на кнопку  и выберите оба объекта. Выбрать оба объекта можно последовательно щелкнув по ним с нажатой клавишей **Shift**.
3. В меню выберите **Контур** → **Сумма**.

Разность

Эта команда удаляет у объекта области, перекрываемые вышележащим выделенным объектом (или объектами). Может применяться только к двум объектам. Алгоритм выполнения операции для двух объектов приведен ниже.

1. Создайте две фигуры с помощью инструментов.
2. Нажмите на кнопку  и выберите оба объекта. Выбрать оба объекта можно последовательно щелкнув по ним с нажатой клавишей **Shift**.
3. В меню выберите **Контур** → **Разность**.

Пересечение

Эта команда позволяет создать новый объект, который включает в себя область пересечения двух или более объектов, выделенных перед слиянием. Если выделено более двух объектов, то необходимо, чтобы во всех выделенных объектах был пересекающийся сегмент. Если такого сегмента нет, то команда игнорируется. Пересекающийся фрагмент может быть только один. Получаемый в результате выполнения операции объект всегда использует настройки стиля (заливки и штриха) нижнего объекта. Алгоритм выполнения операции для двух объектов приведен ниже.

1. Создайте две фигуры с помощью инструментов.
2. Нажмите на кнопку  и выберите оба объекта. Выбрать оба объекта можно последовательно щелкнув по ним с нажатой клавишей **Shift**.
3. В меню выберите **Контур** → **Пересечение**.

Исключающее ИЛИ

Эта команда делает пересекающиеся области прозрачными. Может применяться только к двум объектам. Алгоритм выполнения операции для двух объектов приведен ниже.

1. Создайте две фигуры с помощью инструментов.
2. Нажмите на кнопку  и выберите оба объекта. Выбрать оба объекта можно последовательно щелкнув по ним с нажатой клавишей **Shift**.
3. В меню выберите **Контур** → **Исключающее ИЛИ**.

Разделить

Данная команда соединяет в себя команды **Разность** и **Пересечение**. Число выделенных объектов не может быть больше двух. Алгоритм выполнения операции для двух объектов приведен ниже.

1. Создайте две фигуры с помощью инструментов.
2. Нажмите на кнопку  и выберите оба объекта. Выбрать оба объекта можно последовательно щелкнув по ним с нажатой клавишей **Shift**.
3. В меню выберите **Контур** → **Разделить**.

Работа с узлами

Инструменты для управления узлами

Инструмент для управления узлами  предназначен для редактирования и выбора узлов. Что бы активизировать инструмент для управления узлами, можно использовать боковое окно панели инструментов, этот инструмент расположен в нем вторым сверху или нажать клавишу **F2**. При этом изменится состав кнопок контекстной панели инструментов. Она станет выглядеть так, как показано на рис. 5.1.



Рис. 5.1. Панель инструментов управления узлами

Для того чтобы на активной фигуре выделить узлы, необходимо выбрать кнопку «Преобразовать выбранный объект в контур»  или нажать комбинацию клавиш **Shift+Ctrl+C**.

Если инструмент управления узлами активен, то по контуру активной фигуры могут отображаться узлы в виде квадратиков. Для того чтобы выбрать узел, просто щелкните по нему. Щелчок по контуру между узлами выбирает оба этих узла. Если вы хотите добавить или удалить узел удерживайте при щелчке мыши клавишу **Shift**. Так же добавить узел можно с помощью двойного щелчка левой кнопкой мыши по контуру активной фигуры.

Если необходимо выбрать все узлы фигуры, то можно воспользоваться комбинацией клавиш **Ctrl + A** в этом случае будут выбраны все узлы кроме вложенных. Для того чтобы выбрать все узлы, включая вложенные, следует использовать комбинацию клавиш **Ctrl + Alt + A**.

Для изменения формы можно использовать следующие кнопки:



Клавиша  показывает или скрывает обрисовку контура.

Перемещение узлов

Перемещать узлы можно с помощью мыши обычным образом. Если удерживать при перемещении узла клавишу **Ctrl**, то узел сможет перемещаться только по вертикали или по горизонтали. Удерживая комбинацию клавиш **Ctrl+Alt**, можно перемещать узел строго вдоль его направляющей.

Перемещать узлы можно также с помощью стрелок на клавиатуре. В этом случае объект будет перемещаться с шагом 2 пикселя (по умолчанию, но эту настройку можно изменить).

После того как узел выбран, если это возможно для данного вида узла, то будет отображаться его направляющая. Расположение направляющей также влияет на вид кривой этого узла. С помощью маркеров на концах направляющей можно изменять ее длину и вращать ее. Удерживая при вращении направляющей клавишу **Ctrl**, можно вращать ее с интервалом 15 градусов. Удержание клавиши **Alt** блокирует изменение длины направляющей. Клавиша **Shift** позволяет перемещать обе направляющих.

Горячие клавиши

Shift удерживайте клавишу для выбора нескольких узлов.

Двойной щелчок или **Shift+Alt** на пути или кривой для создания нового узла. Создает новый узел, не изменяя форму контура.

TAB выбирает следующий узел.

Shift +Tab выбирает предыдущий узел.

Ctrl+Alt удаляет узел.

Ctrl + щелчок указателем мыши на маркер направляющей обнуляет ее длину. Что бы вытянуть направляющую назад из узла используйте клавишу **Shift**.

Задание по лабораторной работе

Используя описанные в данном пособии приемы, создайте векторное изображение по образцу из списка ниже, соответствующему номеру вашего варианта (взять у преподавателя).

Проанализируйте проделанную работу в выводе.

Варианты заданий:

	
1	2



3



4



5



6



7



8



9



10



11



12



13



14



15



16

	
17	18
	
19	20
	
21	22
	
23	24

6. ЛАБОРАТОРНАЯ РАБОТА «ПРОГРАММИРОВАНИЕ ГРАФИКИ»

Цель лабораторной работы: изучить возможности Visual Studio по созданию простейших графических изображений. Написать и отладить программу построения на экране различных графических примитивов.

Сообщение WM_PAINT

Прежде чем приступить к описанию способов рисования в окнах, применяемых приложениями .NET Frameworks, расскажем о том, как это делают «классические» приложения Microsoft Windows.

ОС Microsoft Windows следит за перемещением и изменением размера окон и при необходимости извещает приложения, о том, что им следует перерисовать содержимое окна. Для извещения в очередь приложения записывается сообщение с идентификатором **WM_PAINT**. Получив такое сообщение, функция окна должна выполнить перерисовку всего окна или его части, в зависимости от дополнительных данных, полученных вместе с сообщением **WM_PAINT**.

Для облегчения работы по отображению содержимого окна весь вывод в окно обычно выполняют в одном месте приложения – при обработке сообщения **WM_PAINT** в функции окна. Приложение должно быть сделано таким образом, чтобы в любой момент времени при поступлении сообщения **WM_PAINT** функция окна могла перерисовать все окно или любую его часть, заданную своими координатами.

Последнее нетрудно сделать, если приложение будет хранить где-нибудь в памяти свое текущее состояние, пользуясь которым функция окна сможет перерисовать окно в любой момент времени.

Здесь не имеется в виду, что приложение должно хранить образ окна в виде графического изображения и восстанавливать его при необходимости, хотя это и можно сделать. Приложение должно хранить информацию, на основании которой оно может в любой момент времени перерисовать окно.

Сообщение **WM_PAINT** передается функции окна, если стала видна область окна, скрытая раньше другими окнами, если пользователь изменил размер окна или выполнил операцию прокрутки изображения в окне. Приложение может передать функции окна сообщение **WM_PAINT** явным образом, вызывая функции программного интерфейса Win32 API, такие как **UpdateWindow**, **InvalidateRect** или **InvalidateRgn**.

Иногда ОС Microsoft Windows может сама восстановить содержимое окна, не посылая сообщение **WM_PAINT**. Например, при перемещении курсора мыши или значка свернутого приложения ОС восстанавливает содержимое окна. Если же ОС не может восстановить окно, функция окна получает от ОС сообщение **WM_PAINT** и перерисовывает окно самостоятельно.

Событие Paint

Для форм класса **System.Windows.Forms** предусмотрен удобный объектно-ориентированный способ, позволяющий приложению при необходимости перерисовывать окно формы в любой момент времени. Когда вся клиентская область окна формы или часть этой области требует перерисовки, форме передается событие **Paint**. Все, что требуется от программиста, это создать обработчик данного события, наполнив его необходимой функциональностью.

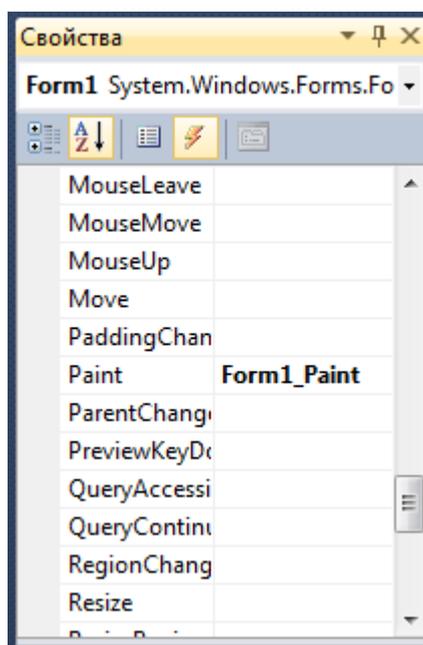


Рис. 6.1. Создание обработчика события Paint

Объект Graphics для рисования

Перед тем как рисовать линии и фигуры, отображать текст, вывести изображения и управлять ими в GDI¹ необходимо создать объект **Graphics**. Объект Graphics представляет поверхность рисования GDI и

¹ GDI (Graphics Device Interface) – один из основных компонентов операционной системы, предоставляющий средства для обработки двумерной графики, рисунков и для решения типографских задач.

используется для создания графических изображений. Ниже представлены два этапа работы с графикой.

1. Создание объекта **Graphics**.
2. Использование объекта **Graphics** для рисования линий и фигур, отображения текста или изображения и управления ими.

Существует несколько способов создания объектов **Graphics**. Одним из самых используемых является получение ссылки на объект **Graphics** через объект **PaintEventArgs** при обработке события **Paint** формы или элемента управления:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics; // Объявляется объект Graphics
    // Далее вставляется код рисования
}
```

Методы и свойства класса **Graphics**

Имена большого количества методов, определенных в классе **Graphics**, начинаются с префикса **Draw*** и **Fill***. Первые из них предназначены для рисования текста, линий и не закрашенных фигур (таких, например, как прямоугольные рамки), а вторые – для рисования закрашенных геометрических фигур. Мы рассмотрим применение только самых важных из этих методов, а полную информацию Вы найдете в документации.

Метод **DrawLine** рисует линию, соединяющую две точки с заданными координатами². Ниже мы привели прототипы различных перегруженных версий этого метода:

```
public void DrawLine(Pen, Point, Point);
public void DrawLine(Pen, PointF, PointF);
public void DrawLine(Pen, int, int, int, int);
public void DrawLine(Pen, float, float, float, float);
```

Первый параметр задает инструмент для рисования линии – перо. Перья создаются как объекты класса **Pen**, например:

```
Pen p = new Pen(Brushes.Black, 2);
```

Здесь мы создали черное перо толщиной 2 пиксела. Создавая перо, Вы можете выбрать его цвет, толщину и тип линии, а также другие атрибуты.

² Начало координат находится в левом верхнем углу элемента, на котором происходит прорисовка. В нашем случае в левом верхнем углу окна. Ось *y* направлена вниз, ось *x* направлена направо.

Остальные параметры перегруженных методов **DrawLine** задают координаты соединяемых точек. Эти координаты могут быть заданы как объекты класса **Point** и **PointF**, а также в виде целых чисел и чисел с плавающей десятичной точкой.

В классах **Point** и **PointF** определены свойства **X** и **Y**, задающие, соответственно, координаты точки по горизонтальной и вертикальной оси. При этом в классе **Point** эти свойства имеют целочисленные значения, а в классе **PointF** – значения с плавающей десятичной точкой.

Третий и четвертый вариант метода **DrawLine** позволяет задавать координаты соединяемых точек в виде двух пар чисел. Первая пара определяет координаты первой точки по горизонтальной и вертикальной оси, а вторая – координаты второй точки по этим же осям. Разница между третьим и четвертым методом заключается в использовании координат различных типов (целочисленных **int** и с плавающей десятичной точкой **float**).

Чтобы испытать метод **DrawLine** в работе, создайте приложение **DrawLineApp** (аналогично тому, как Вы создавали предыдущее приложение). В этом приложении создайте следующий обработчик события **Paint**:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    g.Clear(Color.White);
    for (int i = 0; i < 50; i++)
    {
        g.DrawLine(new Pen(Brushes.Black, 2), 10, 4 * i + 20, 200, 4 * i + 20);
    }
}
```

Здесь мы вызываем метод **DrawLine** в цикле, рисуя 50 горизонтальных линий (рис. 6.2).

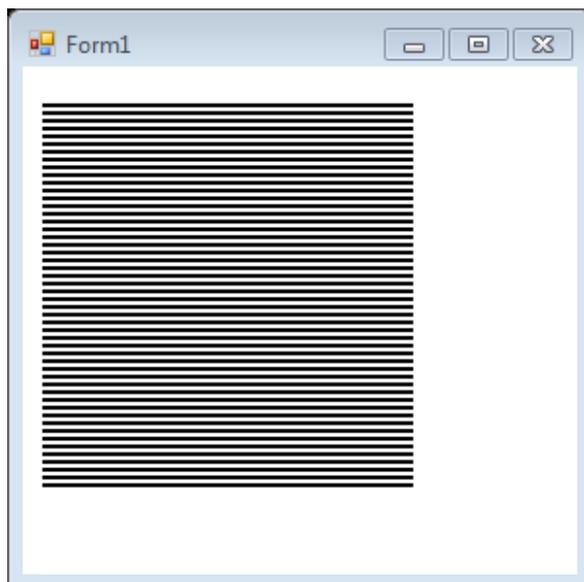


Рис. 6.2. Пример использования *DrawLine*

Вызвав один раз метод `DrawLines`, можно нарисовать сразу несколько прямых линий, соединенных между собой. Иными словами, метод `DrawLines` позволяет соединить между собой несколько точек. Координаты этих точек по горизонтальной и вертикальной оси передаются методу через массив класса `Point` или `PointF`:

```
public void DrawLines(Pen, Point[]);  
public void DrawLines(Pen, PointF[]);
```

Для демонстрации возможностей метода `DrawLines` создайте приложение. Создайте кисть `pen` для рисования линий:

```
Pen pen =new Pen(Color.Black, 2);
```

а также массив точек `points`, которые нужно соединить линиями:

```
Point[] points =new Point[50];  
  
for(int i =0; i < 20; i++)  
  {  
    int xPos;  
    if(i%2 ==0)  
    {  
      xPos =10;  
    }  
    else  
    {  
      xPos =400;  
    }  
    points[i] =new Point(xPos, 10 * i);  
  }
```

Код будет выглядеть следующим образом:

```
public partial class Form1 : Form  
{  
  Point[] points = new Point[50];  
  Pen pen = new Pen(Color.Black, 2);  
  
  public Form1()  
  {  
    InitializeComponent();  
  }  
  
  private void Form1_Paint(object sender, PaintEventArgs e)  
  {  
    Graphics g = e.Graphics;  
    g.DrawLines(pen, points);  
  }
```

```

private void Form1_Load(object sender, EventArgs e)
{
    for (int i = 0; i < 20; i++)
    {
        int xPos;
        if (i % 2 == 0)
        {
            xPos = 10;
        }
        else
        {
            xPos = 400;
        }
        points[i] = new Point(xPos, 10 * i);
    }
}
}

```

Результат приведен на рис. 6.3.

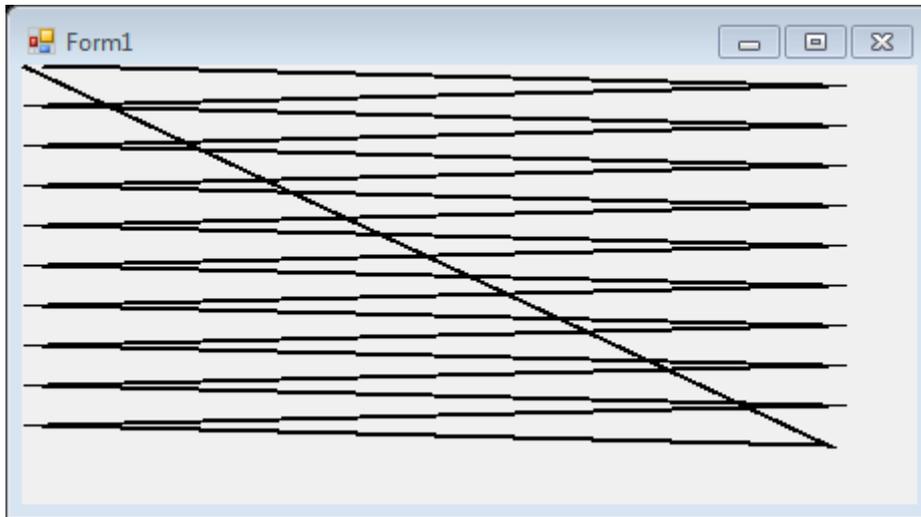


Рис. 6.3. Пример использования массива точек

Для прорисовки прямоугольников можно использовать метод **DrawRectangle(Pen, int, int, int, int)**; В качестве первого параметра передается перо класса Pen. Остальные параметры задают расположение и размеры прямоугольника.

Для прорисовки многоугольников можно использовать метод **DrawPolygon(Pen, Point[])**;

Метод **DrawEllipse** рисует эллипс, вписанный в прямоугольную область, расположение и размеры которой передаются ему в качестве параметров. При помощи метода **DrawArc** программа может нарисовать сегмент эллипса. Сегмент задается при помощи координат прямоугольной области, в которую вписан эллипс, а также двух углов, отсчитываемых в направлении против часовой стрелки. Первый угол Angle1 задает

расположение одного конца сегмента, а второй Angle2 – расположение другого конца сегмента (рис. 6.4).

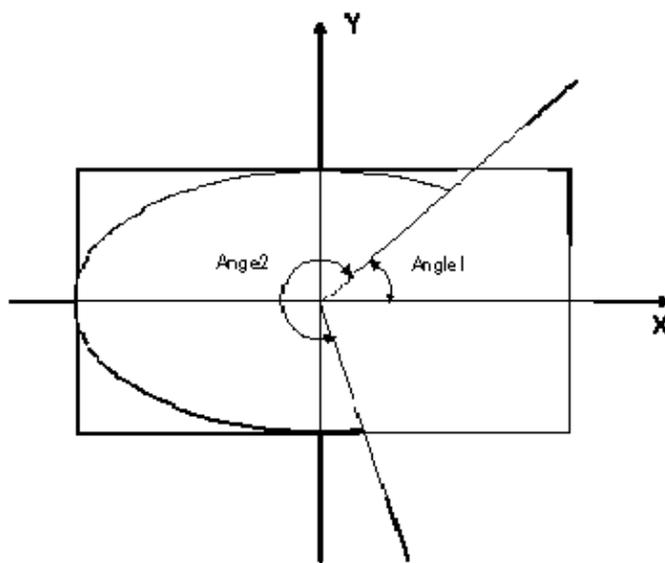


Рис. 6.4. Углы и прямоугольник, задающие сегмент эллипса

В классе **Graphics** определен ряд методов, предназначенных для рисования закрашенных фигур. Имена некоторых из этих методов, имеющих префикс **Fill**:

FillRectangle (рисование закрашенного прямоугольника), **FillRectangles** (рисование множества закрашенных многоугольников), **FillPolygon** (рисование закрашенного многоугольника), **FillEllipse** (рисование закрашенного эллипса) **FillPie** (рисование закрашенного сегмента эллипса) **FillClosedCurve** (рисование закрашенного сплайна) **FillRegion** (рисование закрашенной области типа **Region**).

Есть два отличия методов с префиксом **Fill** от одноименных методов с префиксом **Draw**. Прежде всего, методы с префиксом **Fill** рисуют закрашенные фигуры, а методы с префиксом **Draw** – не закрашенные. Кроме этого, в качестве первого параметра методам с префиксом **Fill** передается не перо класса **Pen**, а кисть класса **SolidBrush**. Ниже приведем пример выводющий закрашенный прямоугольник:

```
SolidBrush B=new SolidBrush(Color.DeepPink);  
g.FillRectangle(B,0,0,100,100);
```

Как видите платформа .Net содержит большое число классов со многими методами и свойствами. Нет смысла описывать все классы, методы в каком либо учебнике или в данном пособии, поскольку по любому методу или классу можно получить MSDN справку набрав наименование метода в среде Visual Studio и нажав на нем клавишу F1. Также,

при наборе метода в редакторе кода среда показывает краткую справку о передаваемых параметрах.

Задание по лабораторной работе

Изучите с помощью справки MSDN методы и свойства классов **Graphics**, **Color**, **Pen** и **SolidBrush**. Создайте собственное приложение выводящий на форму рисунок, состоящий из различных объектов (линий, многоугольников, эллипсов, прямоугольников и пр.), не закрашенных и закрашенных полностью. Используйте разные цвета и стили линий (сплошные, штриховые, штрих-пунктирные).

7. ЛАБОРАТОРНАЯ РАБОТА «ПРОСТЕЙШАЯ АНИМАЦИЯ»

Цель лабораторной работы: изучить возможности Visual Studio по созданию простейшей анимации. Написать и отладить программу, выводящую на экран анимационное изображение.

Работа с таймером

Класс для работы с таймером (Timer) формирует в приложении повторяющиеся события. События повторяются с периодичностью, указанной в миллисекундах, в свойстве **Interval**. Установка свойства **Enabled** в значение **true** запускает таймер. Каждый тик таймера порождает событие **Tick**, обработчик которого обычно и создают в приложении. В этом обработчике могут изменяться какие-либо величины, и вызываться принудительная перерисовка окна. Напоминаем, что вся отрисовка при создании анимации должна находиться в обработчике события **Paint**.

Создание анимации

Для создания простой анимации достаточно использовать таймер, при тике которого будут изменяться параметры изображения (например, координаты концов отрезка) и обработчик события **Paint** для рисования по новым параметрам. При таком подходе не надо заботиться об удалении старого изображения (как в идеологии MS DOS), ведь оно создается в окне заново.

В качестве примера рассмотрим код анимации секундной стрелки часов:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        //описываем переменные доступные в любом обработчике событий класса
        private int x1, y1, x2, y2, r;
        private double a;
        private Pen pen = new Pen(Color.DarkRed, 2);
    }
}
```

```

public Form1()
{
    InitializeComponent();
}

private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    g.DrawLine(pen, x1, y1, x2, y2); //рисует секундную стрелку
}

private void Form1_Load(object sender, EventArgs e)
{
    //определяем центр экрана
    x1 = ClientSize.Width / 2;
    y1 = ClientSize.Height / 2;
    r = 150; //задаем радиус
    a = 0; //задаем угол поворота
    //определяем конец часовой стрелки с учетом центра экрана
    x2 = x1 + (int)(r * Math.Cos(a));
    y2 = y1 - (int)(r * Math.Sin(a));
}

private void timer1_Tick(object sender, EventArgs e)
{
    a -= 0.1; //уменьшаем угол на 0,1 радиану
    //определяем конец часовой стрелки с учетом центра экрана
    x2 = x1 + (int)(r * Math.Cos(a));
    y2 = y1 - (int)(r * Math.Sin(a));
    Invalidate(); //вынудительный вызов перерисовки (Paint)
}
}
}

```

Движение по траектории

Движение по траектории реализуется аналогично выше рассмотренному примеру. Для реализации движения по прямой приращиваются на определённые константы переменные, являющиеся узловыми точками (в примере переменные x_2 , y_2). Для задания более сложной траектории можно использовать различные параметрические кривые. В случае движения на плоскости обычно изменяется один параметр. Рассмотрим пример реализации движения окружности по декартову листу.

Декартов лист – плоская кривая третьего порядка, удовлетворяющая уравнению в прямоугольной системе $x^3 + y^3 = 3axy$. Параметр $3a$ определяется как диагональ квадрата, сторона которого равна наибольшей хорде петли.

При переходе к параметрическому виду получаем:

$$\begin{cases} x = \frac{3at}{1+t^3} \\ y = \frac{3at^2}{1+t^3} \end{cases}$$

где $t = \operatorname{tg} \varphi$.

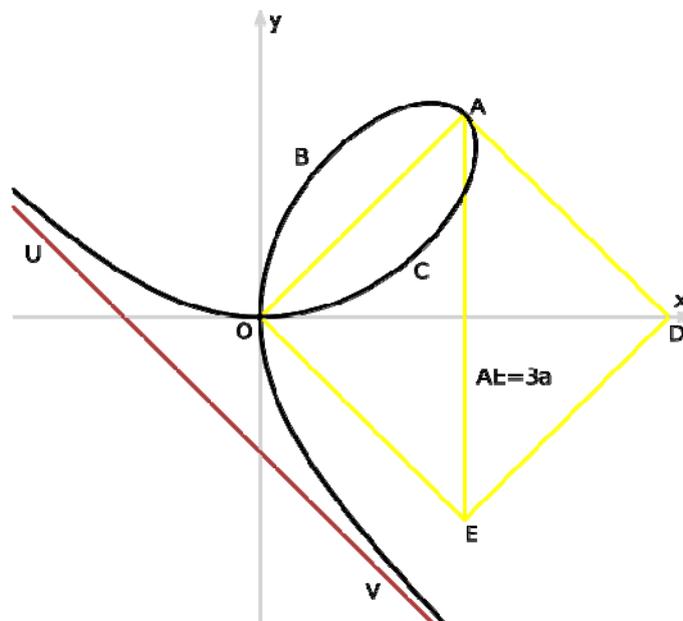


Рис. 7.1. Декартов лист

Программная реализация выглядит следующим образом:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        private int x1, y1, x2, y2;
        private double a, t, fi;
        private Pen pen = new Pen(Color.DarkRed, 2);

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            x1 = ClientSize.Width / 2;
            y1 = ClientSize.Height / 2;
            a = 150;
            fi = -0.5;
            t = Math.Tan(fi);
            x2 = x1 + (int)((3 * a * t) / (1 + t * t * t));
            y2 = y1 - (int)((3 * a * t * t) / (1 + t * t * t));
        }
    }
}

```

```

private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    g.DrawEllipse(pen, x2, y2, 20, 20);
}

private void timer1_Tick(object sender, EventArgs e)
{
    fi += 0.01;
    t = Math.Tan(fi);
    x2 = x1 + (int)((3 * a * t) / (1 + t * t * t));
    y2 = y1 - (int)((3 * a * t * t) / (1 + t * t * t));
    Invalidate();
}

private void button1_Click(object sender, EventArgs e)
{
}
}
}

```

Описание ряда интересных кривых для создания траектории движения можно найти в «Википедии» в статье «Циклоидальная кривая».

Задание по лабораторной работе

Изучите с помощью справки MSDN методы и свойства классов **Graphics**, **Color**, **Pen** и **SolidBrush**. Создайте собственное приложение для **анимации** в соответствии с индивидуальным заданием.

1. Создайте программу движения окружности по **циклоиде**.
2. Разработайте программу движения окружности по **гипоциклоиде** при $k = 3$, $k = 4$, $k = 6$, $k = 2,1$, $k = 5,5$
3. Создайте программу движения окружности по **эпициклоиде** при различных значениях k .
4. Разработайте программу отображающую процесс построения **гипотрохида**.
5. Создайте программу моделирующую построение кривых с помощью **спирографа**. R , r , d задаются произвольно.
6. Создайте программу, показывающую движение окружности по **синусоиде**.
7. Создайте приложение, отображающее движение окружности по **спирали**.
8. Разработайте программу движения окружности по **трактрисе** (кривой погони).
9. Создайте программу, показывающую движение окружности по **трисектрисе Каталана** (Кубика Чирнгауза).

10. Разработайте программу, отображающую процесс построения **фигур Лиссажу**, с произвольными задаваемыми параметрами.
11. Разработайте приложение, отображающее процесс построения **связанных звезд**, с произвольным числом вершин.
12. Создайте программу, отображающую движения **маятника** с затуханием.
13. Создайте программу анимирующую процесс построения различных **спиралей** (параболических, логарифмических, архимедовых спираль Корню, клотоида).
14. Разработайте программу, отображающую процесс построения **Лемнискаты Бернулли**.
15. Создайте программу движения объекта по **кривой Персея** при различных значениях a , b и c .
16. Разработайте программу движения точки вдоль **кривой Безье** четвертого порядка. Узловые точки задаются произвольно пользователем до построения кривой.
17. Разработайте программу **анимации падения снежинки**, которые падают по разным траекториям и с разными скоростями.
18. Разработайте программу **анимации летающего бумеранга**.
19. Создайте программу, показывающую **падение нескольких звезд** одновременно.
20. Создайте приложение, отображающее **хаотичное движение** звезды в окне.
21. Создайте программу, показывающую **движение окружности вдоль многоугольника**. Число вершин вводится пользователем до анимации.
22. Создайте приложение, отображающее **броуновское движение** молекулы в окне.
23. Разработайте программу **анимации движения планет** в солнечной системе.
24. Создайте программу, показывающую движение квадратика по траектории, состоящей из 100 точек, и хранящихся в специальном массиве.

8. ЛАБОРАТОРНАЯ РАБОТА «РАБОТА С РАСТРОВЫМИ ИЗОБРАЖЕНИЯМИ»

Цель лабораторной работы: изучить возможности Visual Studio по открытию и сохранению файлов. Написать и отладить программу для создания или обработки изображений.

Отображение графических файлов

Обычно для отображения точечных рисунков, рисунков из мета-файлов, значков, рисунков из файлов в формате BMP, JPEG, GIF или PNG используется объект **PictureBox**, т. е. элемент управления **PictureBox** действует как контейнер для картинок. Можно выбрать изображение для вывода, присвоив значение свойству **Image**. Свойство **Image** может быть установлено в окне **Свойства** или в коде программы, указывая на рисунок, который следует отображать.

Элемент управления **PictureBox** содержит и другие полезные свойства, в том числе: **AutoSize** определяющее, будет ли изображение растянуто в элементе **PictureBox**, и **SizeMode**, которое может использоваться для растягивания, центрирования или увеличения изображения в элементе управления **PictureBox**.

Перед добавлением рисунка к элементу управления **PictureBox** в проект обычно добавляется файл рисунка в качестве ресурса³. После добавления ресурса к проекту можно повторно использовать его. Например, может потребоваться отображение одного и того же изображения в нескольких местах.

Необходимо отметить, что поле **Image** само является классом для работы с изображениями, у которого есть свои методы. Например, метод **FromFile** используется для загрузки изображения из файла. Кроме класса **Image** существует класс **Bitmap**, который расширяет возможности класса **Image** за счет дополнительных методов для загрузки, сохранения и использования растровых изображений. Так метод **Save** класса **Bitmap** позволяет сохранять изображения в разных форматах, а методы **GetPixel** и **SetPixel** позволяют получить доступ к отдельным пикселям рисунка.

³ В приложениях Visual C# часто содержатся данные, не являющиеся исходным кодом. Такие данные называются ресурсами проекта и могут включать двоичные данные, текстовые файлы, аудио- и видеофайлы, таблицы строк, значки, изображения, XML-файлы или любой другой тип данных, необходимых для приложения. Данные ресурсов проекта хранятся в формате XML в файле с расширением RESX (имя по умолчанию – Resources.resx), который можно открыть в Обозревателе решений.

Компоненты OpenFileDialog и SaveFileDialog

Компонент **OpenFileDialog** является стандартным диалоговым окном. Он аналогичен диалоговому окну «Открыть файл» операционной системы Windows. Компонент **OpenFileDialog** позволяет пользователям просматривать папки личного компьютера или любого компьютера в сети, а также выбирать файлы, которые требуется открыть. Для вызова диалогового окна для выбора файла можно использовать метод **ShowDialog()** который возвращает **true** при корректном выборе.

Диалоговое окно возвращает путь и имя файла, который был выбран пользователем в специальном свойстве **FileName**.

Простой графический редактор

Создайте приложение, реализующее простой графический редактор. Функциями этого редактора должны быть: открытие рисунка, рисование поверх него простой кистью, сохранение рисунка в другой файл. Для этого создайте форму и разместите на ней элементы управления **button** и **picturebox** (рис. 7.1).

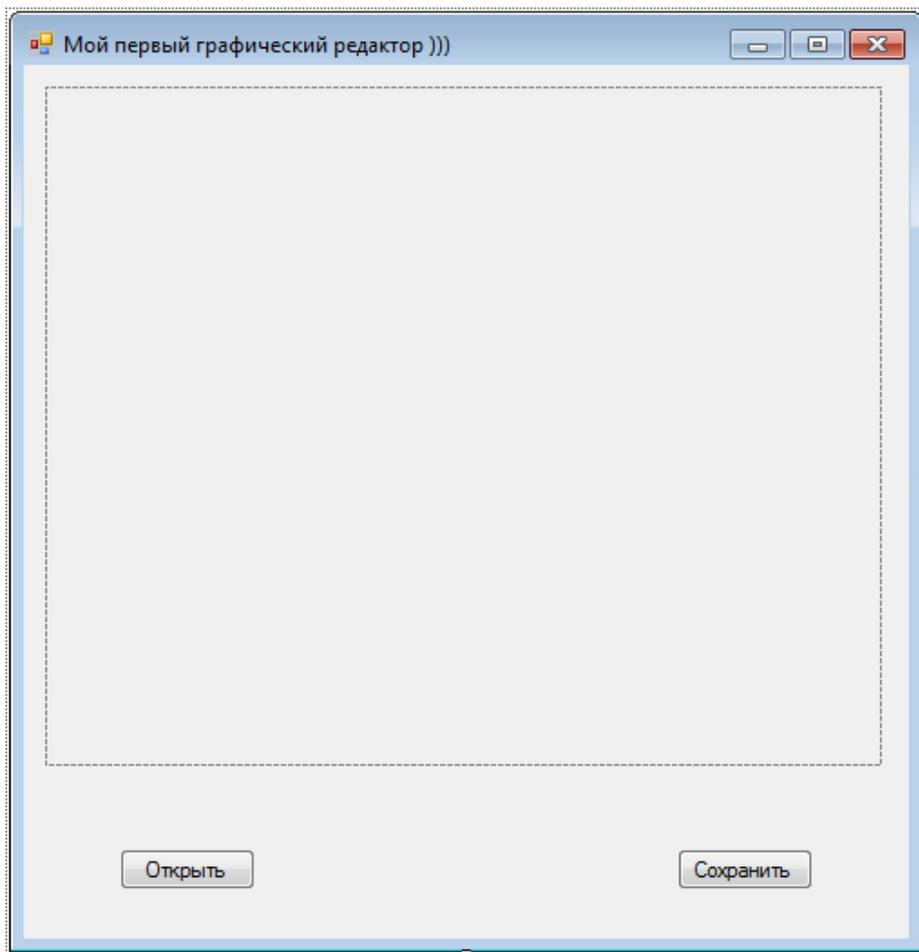


Рис. 8.1. Форма для графического редактора

В этом случае на понадобится из панели элементов размещать на форме компоненты диалоговых окон OpenFileDialog и SaveFileDialog. Эти элементы будут порождены динамически в ходе выполнения программы с помощью конструктора. Например так:

```
OpenFileDialog dialog = new OpenFileDialog();
```

Далее они будут вызываться с помощью метода **ShowDialog()**.

Для кнопок «Открыть» и «Сохранить» создайте свои обработчики события. Также создайте обработчик события **Load** для формы. Для элемента управления **picturebox1** создайте обработчики события **MouseDown**, **MouseMove**. Код приложения будет выглядеть следующим образом:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        //Объявляем переменные доступные в каждом обработчике события
        private Point PreviousPoint, point; //Точка до перемещения курсора мыши
                                           //и текущая точка

        private Bitmap bmp;
        private Pen blackPen;
        private Graphics g;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            blackPen = new Pen(Color.Black, 4); //подготавливаем перо
        }

        private void button1_Click(object sender, EventArgs e)
        {
            //открытие файла
            OpenFileDialog dialog = new OpenFileDialog();
            //задаем расширения файлов
            dialog.Filter = "Image files (*.BMP, *.JPG, *.GIF, *.TIF, *.PNG,
*.ICO, *.EMF, *.WMF)|*.bmp;*.jpg;*.gif;*.tif;*.png;*.ico;*.emf;*.wmf";
            if (dialog.ShowDialog() == DialogResult.OK) //вызываем диалоговое окно
            {
                Image image = Image.FromFile(dialog.FileName); //Загружаем в
image
                //изображение из выбранного файла

                int width = image.Width;
                int height = image.Height;
            }
        }
    }
}
```

```

        pictureBox1.Width = width;
        pictureBox1.Height = height;

        bmp = new Bitmap(image, width, height); //создаем и загружаем из
                                                //image изображение в формате bmp

        pictureBox1.Image = bmp; //записываем изображение в формате bmp
                                //в pictureBox1
        g = Graphics.FromImage(pictureBox1.Image); //подготавливаем
                                                //Graphics для рисования в
pictureBox1
    }
}

private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
{ // обработчик события нажатия кнопки на мыши
  // записываем в предыдущую точку (PreviousPoint) текущие координаты
  PreviousPoint.X = e.X;
  PreviousPoint.Y = e.Y;
}

private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
{//Обработчик события перемещения мыши по pictureBox1
  if (e.Button == MouseButtons.Left) //Проверяем нажатие левой
кнопка
  { //запоминаем в point текущее положение курсора мыши
    point.X = e.X;
    point.Y = e.Y;

    //соединяем линией предыдущую точку с текущей
    g.DrawLine(blackPen, PreviousPoint, point);

    //текущее положение курсора мыши сохраняем в PreviousPoint
    PreviousPoint.X = point.X;
    PreviousPoint.Y = point.Y;
    pictureBox1.Invalidate();//Принудительно вызываем перерисовку
  }
}

private void button2_Click(object sender, EventArgs e)
{ //сохранение файла
  SaveFileDialog savedialog = new SaveFileDialog();
  //задаем свойства для savedialog
  savedialog.Title = "Сохранить картинку как ...";
  savedialog.OverwritePrompt = true;
  savedialog.CheckPathExists = true;
  savedialog.Filter =
    "Bitmap File(*.bmp)|*.bmp|" +
    "GIF File(*.gif)|*.gif|" +
    "JPEG File(*.jpg)|*.jpg|" +
    "TIF File(*.tif)|*.tif|" +
    "PNG File(*.png)|*.png";
  savedialog.ShowHelp = true;
  if (savedialog.ShowDialog() == DialogResult.OK)
  {
    // в fileName записываем полный путь к файлу
    string fileName = savedialog.FileName;
    // Убираем из имени три последних символа (расширение файла)

```


Данный код демонстрирует возможность обращения к отдельным пикселям. Цвет каждого пикселя хранится в модели RGB и состоит из трех составляющих: красного, зеленого и синего цвета, называемых каналами. Значение каждого канала может варьироваться в диапазоне от 0 до 255.

Задание по лабораторной работе

Добавьте в приведенный графический редактор свои функции в соответствии с вариантом.

1. Расширьте приложение путем добавления возможности регулировки **яркости и контрастности** как в целом сразу по трём каналам, так и по каждому каналу отдельно.
2. Расширьте приложение путем добавления различных **фильтров размытия** с различным размером ядра.
3. Расширьте приложение путем добавления различных **фильтров повышения контрастности** с различным размером ядра.
4. Расширьте приложение путем добавления различных фильтров **нахождения границ**.
5. Расширьте приложение путем добавления **медианного фильтра**.
6. Создайте функцию, переводящую изображение в черно-белый формат (**реализуйте пороговую фильтрацию**). Пороговое значение задавать с помощью элемента управления TrackBar.
7. Разработайте функцию **построения гистограммы** по трем каналам (R,G,B) вместе и по каждому каналу в отдельности.
8. Реализуйте функцию **поворота** выделенного участка изображения.
9. Реализуйте функцию вертикального и горизонтального **отражения** выделенного участка изображения.
10. Разработайте функцию, оставляющую на изображении только один из каналов (R,G,B). Канал выбирается пользователем.
11. Расширьте приложение путем добавления возможности выбора пользователем цвета и величины кисти.
12. Создайте функцию, выводящую на изображение окружность. Центр окружности совпадает с центром изображения. Все точки вне окружности переводятся в градации серого цвета. Все точки внутри окружности остаются неизменными. Радиус окружности задается пользователем.
13. Создайте функцию, выводящую на изображение треугольник. Для всех точек вне треугольника оставьте только канал **B**. Все точки внутри треугольника переводятся в градации серого цвета.
14. Создайте функцию, выводящую на изображение ромб. Все точки вне ромба переводятся в градации серого цвета. Для всех точек внутри ромба оставьте только канал **G**.

15. Разработайте функцию, которая каждую четную строку изображения переводит в градации серого цвета.
16. Разработайте функцию, которая переводит каждый нечетный столбец пикселей (вертикальные линии) в градации серого цвета.
17. Создайте функцию, разбивающую изображение на четыре равные части. В каждой оставьте значение только одного канала R, G и B, а в четвертой выведите градации серого цвета.
18. Разработайте функцию, заменяющую все точки синего цвета на точки красного цвета.
19. Создайте функцию, **инвертирующую** изображение в градациях серого цвета **в негатив**.
20. Создайте функцию, переводящую изображение в черно-белый формат в соответствии с пороговым значением, которое ввел пользователь. Для анализа используйте только один из каналов (R,G,B).
21. Разработайте функцию для создания эффекта мозаики. При этом изображения разбивается на прямоугольные фрагменты, в каждом из которых выбирается цвет средней точки и этим же цветом закрашивается весь фрагмент.
22. Разработайте функцию, разбивающую изображение на фрагменты, в каждом из которых остается только один из каналов (R, G, B).

9. ЛАБОРАТОРНАЯ РАБОТА «ПРЕОБРАЗОВАНИЯ НА ПЛОСКОСТИ»

Цель лабораторной работы: изучить, как производятся двухмерные преобразования с помощью однородных координат и матрицы преобразования 3×3 . Написать и отладить программу для 2D преобразований.

Простейшие преобразования на плоскости

Рассмотрим преобразования на плоскости.

Для начала заметим, что точки на плоскости задаются с помощью двух ее координат. Таким образом, геометрически каждая точка задается значениями координат вектора относительно выбранной системы координат. Координаты точек можно рассматривать как элементы матрицы $[x, y]$, т. е. в виде вектор-строки или вектор-столбца. Положением этих точек управляют путем преобразования матрицы.

Точки на плоскости x, y можно перенести в новые позиции путем добавления к координатам этих точек констант переноса:

$$[x^* \ y^*] = [x \ y] + [a \ b] = [x+a \ y+b]$$

Рассмотрим результаты матричного умножения матрицы $[x, y]$, определяющей точку P и матрицы преобразований 2×2 общего вида:

$$[x \ y] \cdot \begin{bmatrix} a & b \\ c & d \end{bmatrix} = [(ax+cy)(bx+dy)] = [x^* \ y^*]$$

Проведем анализ полученных результатов, рассматривая x^* и y^* как преобразованные координаты. Для этого исследуем несколько частных случаев.

Рассмотрим случай, когда $a = d = 1$ и $c = b = 0$. Матрица преобразований приводит к матрице, идентичной исходной,

$$[x \ y] \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = [(1x+0y)(0x+1y)] = [x \ y] = [x^* \ y^*]$$

При этом изменений координат точки P не происходит.

Если теперь $d = 1, b = c = 0, a = const$, то:

$$[x \ y] \cdot \begin{bmatrix} a & 0 \\ 0 & 1 \end{bmatrix} = [(ax+0y)(0x+1y)] = [ax \ y] = [x^* \ y^*]$$

Как видно, это приводит к изменению масштаба в направлении x , так как $x^* = ax$. Следовательно, данное матричное преобразование эквивалентно перемещению исходной точки в направлении x .

Теперь положим $b = c = 0$, т. е.:

$$\begin{bmatrix} x & y \end{bmatrix} \cdot \begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix} = [(ax + 0y)(0x + dy)] = [ax \quad dy] = [x^* \quad y^*]$$

В результате получаем изменение масштабов в направлениях x и y . Если $a \neq d$, то перемещения вдоль осей неодинаковы. Если $a = d > 1$, то имеет место увеличение масштаба координат точки P . Если $0 < a = d < 1$, то будет иметь место уменьшение масштаба координат точки P .

Если a или (и) d отрицательны, то происходит отображение координат точек. Рассмотрим это, положив $b = c = 0$, $d = 1$ и $a = -1$, тогда:

$$\begin{bmatrix} x & y \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} = [(-1x + 0y)(0x + 1y)] = [-x \quad y] = [x^* \quad y^*]$$

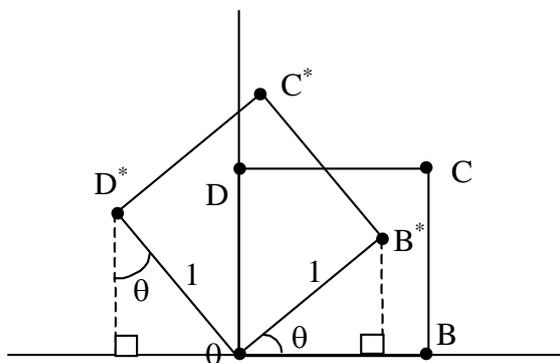
Произошло отображение точки относительно оси y . В случае $b = c = 0$, $a = 1$, $d = -1$, отображение происходит относительно оси x . Если $b = c = 0$, $a = d < 0$, то отображение будет происходить относительно начала координат.

Заметим, что отображение и изменение масштаба вызывают только диагональные элементы матрицы преобразования.

Преобразование общего вида, примененное к началу координат не приведет к изменению координат точки $(0, 0)$. Следовательно, начало координат инвариантно при общем преобразовании. Это ограничение преодолевается за счет использования однородных координат.

Преобразование поворота и отражения

Общую матрицу 2×2 , которая осуществляет вращение фигуры относительно начала координат, можно получить из рассмотрения вращения единичного квадрата вокруг начала координат.



Как следует из рисунка, точка В с координатами (1, 0) преобразуется в точку B^* , для которой $x^* = (1) \cos\theta$ и $y^* = (1) \sin\theta$, а точка D, имеющая координаты (0, 1) переходит в точку D^* с координатами $x^* = (-1) \sin\theta$ и $y^* = (1) \cos\theta$.

Матрица преобразования общего вида записывается так:

$$\begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}.$$

Для частных случаев. Поворот на 90° можно осуществить с помощью матрицы преобразования

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}.$$

Если использовать матрицу координат вершин, то получим, например:

$$\begin{bmatrix} 3 & -1 \\ 4 & 1 \\ 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ -1 & 4 \\ -1 & 2 \end{bmatrix}.$$

Поворот на 180° получается с помощью матрицы $\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$.

В то время как чистое двумерное вращение в плоскости xu осуществляется вокруг оси, перпендикулярной к этой плоскости, отображение определяется поворотом на 180° вокруг оси, лежащей в плоскости xu .

Такое вращение вокруг линии $y = x$ происходит при использовании матрицы $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.

Преобразованные новые выражения определяются соотношением:

$$\begin{bmatrix} 8 & -1 \\ 7 & 3 \\ 6 & 2 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 8 \\ 3 & 7 \\ 2 & 6 \end{bmatrix}.$$

Вращение вокруг $y = 0$ получается при использовании матрицы

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

Однородные координаты

Преобразования переноса, масштабирования и поворота записываются в матричной форме в виде

$$\begin{aligned} P^* &= P + T, \\ P^* &= P \cdot S, \\ P^* &= P \cdot R. \end{aligned}$$

Очевидно, что перенос, в отличие от масштабирования и поворота, реализуется с помощью сложения. Это обусловлено тем, что вводить константы переноса внутрь структуры общей матрицы размера 2×2 не представляется возможным. Желательным является представление преобразований в единой форме – с помощью умножения матриц. Эту проблему можно решить за счет введения третьей компоненты в векторы точек $[x \ y]$ и $[x^* \ y^*]$, т. е. представляя их в виде $[x \ y \ 1]$ и $[x^* \ y^* \ 1]$. Матрица преобразования после этого становится матрицей размера 3×3 , например:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{bmatrix}.$$

Используя эту матрицу, получаем преобразованный вектор $[x^* \ y^* \ 1]$. Добавление третьего элемента к вектору положения и третьего столбца к матрице преобразования позволяет выполнить смещение вектора положения. Третий элемент здесь можно рассматривать как дополнительную координату вектора положения. Итак, вектор положения $[x \ y \ 1]$ при воздействии на него матрицы 3×3 становится вектором положения в общем случае вида $[X \ Y \ H]$. Представленное преобразование было выполнено так, что $[X \ Y \ H] = [x^* \ y^* \ 1]$.

Преобразование, имеющее место в трехмерном пространстве, в нашем случае ограничено плоскостью, поскольку $H = 1$. Если, однако, третий стол-

бец $\begin{bmatrix} p \\ q \\ s \end{bmatrix}$ матрицы преобразования T размера 3×3 отличен от 0, то в резуль-

тате матричного преобразования получим $[x \ y \ 1] \cdot T = [X \ Y \ H]$, где $H \neq 1$.

Плоскость, в которой теперь лежит преобразованный вектор положения, находится в трехмерном пространстве.

Преобразованные обычные координаты получаются за счет **нормализации** однородных координат, т. е.

$$x^* = \frac{X}{H} \text{ и } y^* = \frac{Y}{H}$$

Геометрически все преобразования x и y происходят в плоскости $H = 1$ после нормализации преобразованных однородных координат.

Преимущество введения однородных координат проявляется при использовании матрицы преобразований общего вида порядка 3×3

$$\left[\begin{array}{ccc|c} a & b & p & \\ c & d & q & \\ m & n & s & 1 \end{array} \right]$$

с помощью которой можно выполнять и другие преобразования, такие как смещение, операции изменения масштаба и сдвига, обусловленные матричными элементами a , b , c и d . Указанные операции рассмотрены ранее.

Основная матрица преобразования размера 3×3 для двумерных однородных координат может быть подразделена на четыре части:

$$\left[\begin{array}{cc|c} a & b & p \\ c & d & q \\ \hline m & n & s \end{array} \right]$$

Как мы видим, a , b , c и d осуществляют изменение масштаба, сдвиг и вращение; m и n выполняют смещение, а p и q – получение проекций. Оставшаяся часть матрицы, элемент s , производит полное изменение масштаба. Чтобы показать это, рассмотрим преобразование

$$[X \ Y \ H] = [x \ y \ 1] \cdot \left[\begin{array}{ccc|c} 1 & 0 & 0 & \\ 0 & 1 & 0 & \\ \hline 0 & 0 & s & \end{array} \right] = [x \ y \ s]$$

Здесь $X = x$, $Y = y$, а $H = s$. Это дает $x^* = x/s$ и $y^* = y/s$. В результате преобразования $[x \ y \ 1] \rightarrow [x/s \ y/s \ 1]$ имеет место однородное изменение масштаба вектора положения. При $s < 1$ происходит увеличение, а при $s > 1$ уменьшение масштаба.

Комбинированные преобразования

Рассмотрим комбинированные преобразования на примере поворота вокруг произвольной точки.

Выше было рассмотрено вращение изображения около начала координат. Однородные координаты обеспечивают поворот изображения вокруг точек, отличных от начала координат. В общем случае вращение около произвольной точки может быть выполнено путем переноса центра вращения в начало координат, поворотом относительно начала координат, а затем переносом точки вращения в исходное положение. Таким образом, поворот вектора положения $[x \ y \ 1]$ около точки (m, n) на произвольный угол может быть выполнен с помощью преобразования

$$[x \ y \ 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -m & -n & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{bmatrix} = [X \ Y \ H].$$

Выполнив две операции умножения матриц, можно записать

$$[X \ Y \ H] = [x \ y \ 1] \cdot \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ -m(\cos \theta - 1) + n(\sin \theta) & -m(\sin \theta) - n(\cos \theta - 1) & 1 \end{bmatrix}.$$

Программная реализация

На первом этапе спроектируйте оконный интерфейс. Разместите на форме кнопки для переноса, масштабирования и поворота. При необходимости разместите элементы для ввода значений: например для ввода значений углов поворота. Пример окна приведен на рис. 9.1.

Опишите и задайте начальные значения для матрицы, где будут храниться координаты двумерной фигуры. Назовем эту матрицу матрицей тела и опишем как двумерный массив **Sq** типа **double**. Опишите и задайте как единичную матрицу преобразования 3×3 (двухмерный массив **T** типа **double**). Поскольку массивы **Sq** и **T** будут использоваться повсеместно, то целесообразно задать их в качестве полей класса доступных во всех методах и не тратить время на передачу в качестве параметров.

При нажатии на кнопку должен запускаться процесс, приведённый на рис. 9.2.

Этап задания коэффициентов матрицы преобразования сводится к заданию элементов двумерного массива.

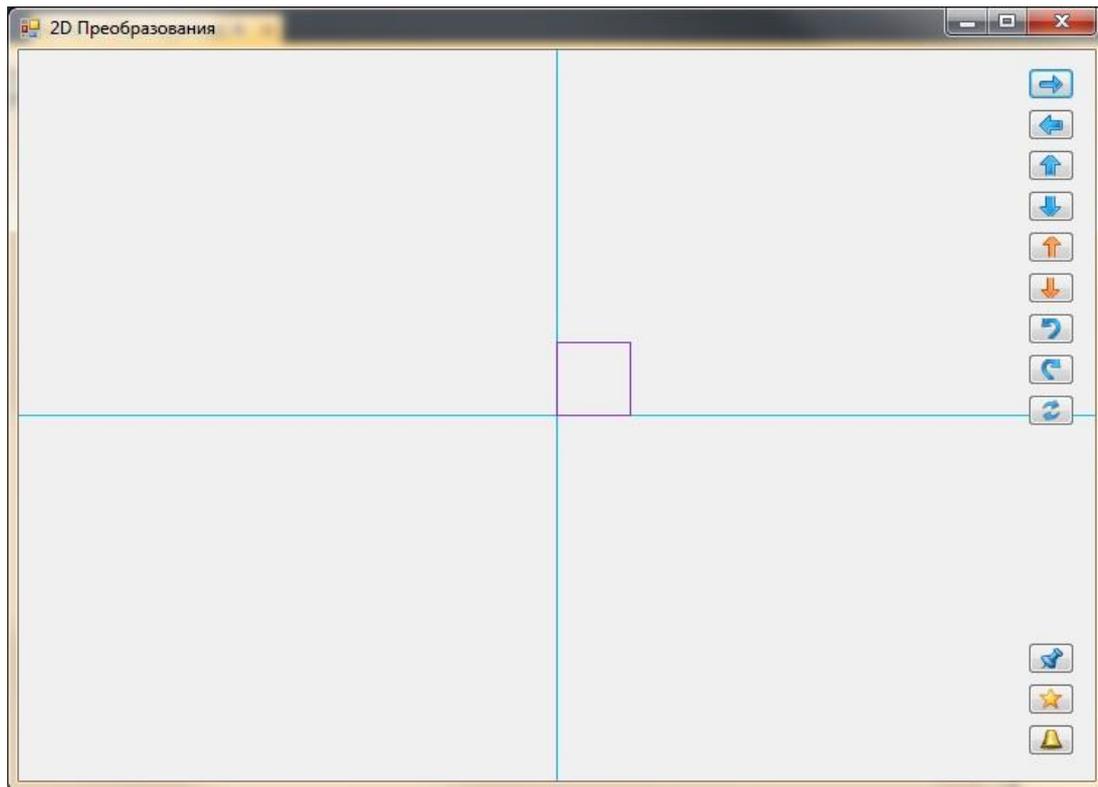


Рис. 9.1. Интерфейс для преобразований на плоскости



Рис. 9.2. Основной процесс визуализации

Как видим из рис. 9.2. на втором этапе выбрана стратегия сохранения изменений при преобразованиях в матрице тела, а не накапливание изменений в матрице преобразований. Исходя из этого тело метода, реализующего данный этап, может выглядеть следующим образом:

```
b = new double[3]; // массив для хранения промежуточных данных
for (int j=0; j<4; j++) // Цикл по вершинам фигуры (4 вершины для квадрата)
{
    for (int i = 0; i < 3; i++)
    {
        b[i] = 0;
        for (int k = 0; k < 3; k++)
            b[i] = b[i] + Sq[j, k]*T[k, i];
    }
    for (int k=0; k<3; k++)
        Sq[j, k]= b[k];
}
}
```

Третий этап нормализации приводит к единице последнюю координату путем деления:

```
for (int j = 0; j < 4; j++)
{
    Sq[j, 0] = Sq[j, 0] / Sq[j, 2];
    Sq[j, 1] = Sq[j, 1] / Sq[j, 2];
    Sq[j, 2] = 1;
}
}
```

Перерисовка должна находиться в обработчике события **Paint**:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics gfx = e.Graphics;
    Pen skyBluePen = new Pen(Brushes.DeepSkyBlue);
    int cx = ClientSize.Width/2;
    int cy = ClientSize.Height/2;
    gfx.DrawLine(skyBluePen, cx, 0, cx, ClientSize.Height);
    gfx.DrawLine(skyBluePen, 0, cy, ClientSize.Width, cy);

    Pen SqPen = new Pen(Brushes.BlueViolet);

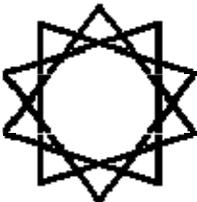
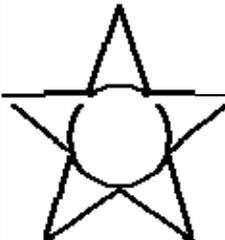
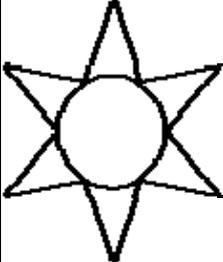
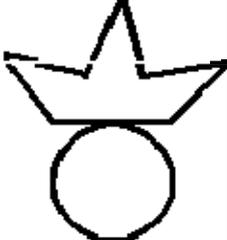
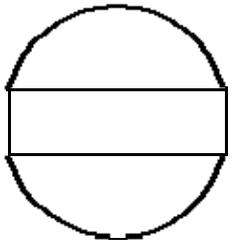
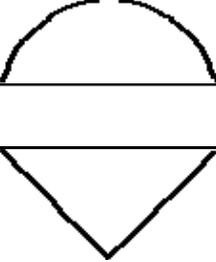
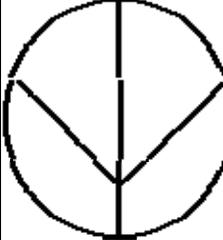
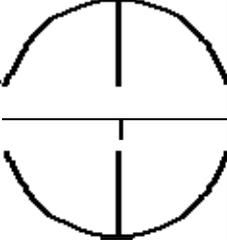
    gfx.DrawLine(SqPen, (int) (cx + Sq[0, 0]), (int) (cy - Sq[0, 1]),
                    (int) (cx + Sq[1, 0]), (int) (cy - Sq[1, 1]));
    gfx.DrawLine(SqPen, (int) (cx + Sq[1, 0]), (int) (cy - Sq[1, 1]),
                    (int) (cx + Sq[2, 0]), (int) (cy - Sq[2, 1]));
    gfx.DrawLine(SqPen, (int) (cx + Sq[2, 0]), (int) (cy - Sq[2, 1]),
                    (int) (cx + Sq[3, 0]), (int) (cy - Sq[3, 1]));
    gfx.DrawLine(SqPen, (int) (cx + Sq[3, 0]), (int) (cy - Sq[3, 1]),
                    (int) (cx + Sq[0, 0]), (int) (cy - Sq[0, 1]));
}
}
```

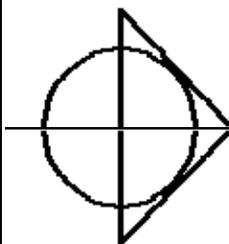
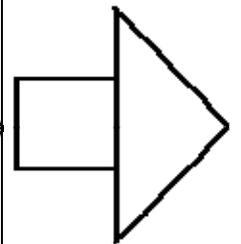
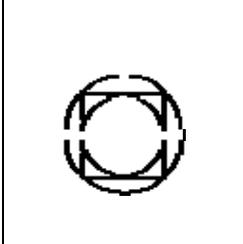
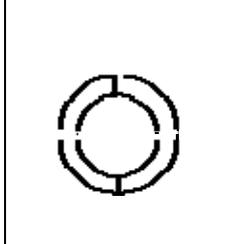
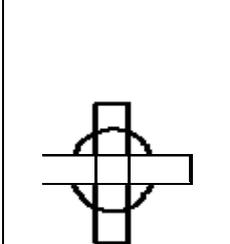
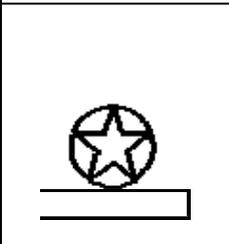
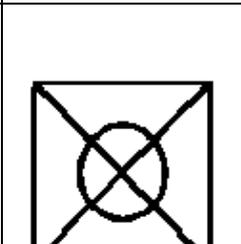
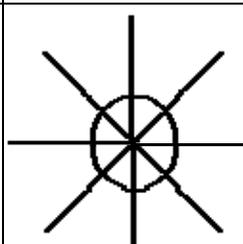
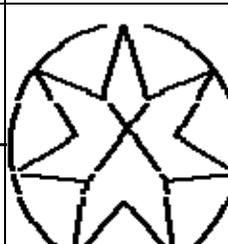
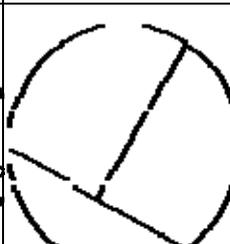
Переменные s_x и s_y это координаты центра окна, используются для прорисовки осей и самой фигуры. Приведение к целому типу, а следовательно округление, производится непосредственно перед прорисовкой и не сохраняется в матрице Sq . Поэтому не будет происходить накопление ошибок округления. Запустите и отладьте программу. По результатам подготовьте отчет.

Задание по лабораторной работе

Реализуйте с заданной совокупностью фигур все виды аффинных преобразований: перенос по оси OX и оси OY , отражение относительно координатных осей и прямой $Y = X$, масштабирование, поворот на заданные углы относительно центра координат и относительно произвольной точки, указываемой в ходе выполнения программы. Предусмотреть восстановление исходной позиции фигур. Управление организовать как через интерфейсные элементы (меню, кнопки, строки редактирования и пр.), так и через «горячие» клавиши.

Варианты заданий

				
Задание № 1	Задание № 2	Задание № 3	Задание № 4	Задание № 5
				
Задание № 6	Задание № 7	Задание № 8	Задание № 9	Задание № 10
				
Задание № 11	Задание № 12	Задание № 13	Задание № 14	Задание № 15

 <p>Задание № 16</p>	 <p>Задание № 17</p>	 <p>Задание № 18</p>	 <p>Задание № 19</p>	 <p>Задание № 20</p>
 <p>Задание № 21</p>	 <p>Задание № 22</p>	 <p>Задание № 23</p>	 <p>Задание № 24</p>	 <p>Задание № 25</p>

10. ЛАБОРАТОРНАЯ РАБОТА «3D ПРЕОБРАЗОВАНИЯ И ПОЛУЧЕНИЕ ПРОЕКЦИЙ»

Цель лабораторной работы: изучить, как производятся преобразования в пространстве. Изучить способ получения проекций с помощью матрицы 4×4 . Написать и отладить программу для 3D преобразований.

Правосторонняя система координат

Рассмотрим трехмерную декартову систему координат, являющуюся **правосторонней**. Примем соглашение, в соответствии с которым будем считать положительными такие повороты, при которых (если смотреть с конца полуоси в направлении начала координат) поворот на 90° против часовой стрелки будет переводить одну полуось в другую. На основе этого соглашения строится следующая таблица, которую можно использовать как для правых, так и для левых систем координат:

Если ось вращения	Положительным будет направление поворота
X	От y к z
Y	От z к x
Z	От x к y

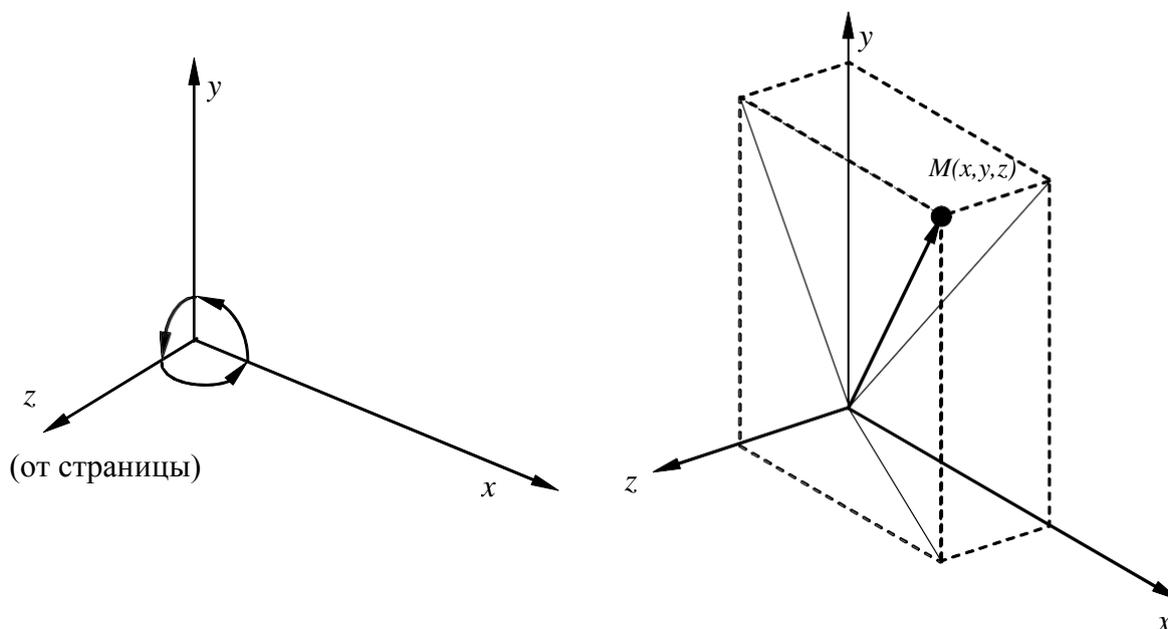


Рис. 10.1. Трехмерная система координат

Преобразования в пространстве

Аналогично тому, как точка на плоскости описывается вектором (x,y) , точка в трехмерном пространстве описывается вектором (x,y,z) .

Как и в двухмерном случае, для возможности реализаций трехмерных преобразований с помощью матриц перейдем к однородным координатам:

$$[x,y,z,1] \text{ или } [X,Y,Z,H]$$

$$[x^*,y^*,z^*,1] = \begin{bmatrix} X & Y & Z \\ H & H & H \end{bmatrix}, \text{ где } H \neq 1, H \neq 0.$$

Обобщенная матрица преобразования 4×4 для трехмерных однородных координат имеет вид

$$T = \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ h & i & j & r \\ l & m & n & s \end{bmatrix}$$

Эта матрица может быть представлена в виде четырех отдельных частей:

$$\begin{bmatrix} 3 \times 3 & 3 \times 1 \\ 1 \times 3 & 1 \times 1 \end{bmatrix}.$$

- Матрица 3×3 осуществляет линейное⁴ преобразование в виде изменения масштаба, сдвига и вращения.
- Матрица 1×3 производит перенос.
- Матрица 3×1 - преобразования в перспективе.
- Скалярный элемент 1×1 выполняет общее изменение масштаба.

Рассмотрим воздействие матрицы 4×4 на однородный вектор $[x,y,z,1]$:

Трехмерный перенос

Трехмерный перенос – является простым расширением двумерного:

$$T(Dx,Dy,Dz) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Dx & Dy & Dz & 1 \end{bmatrix},$$

т. е. $[x,y,z,1] * T(Dx,Dy,Dz) = [x+Dx,y+Dy,z+Dz,1]$.

⁴ **Линейное преобразование** трансформирует исходную линейную комбинацию векторов в некоторую линейную их комбинацию.

Трехмерное изменение масштаба

Рассмотрим **частичное** изменение масштаба. Оно реализуется следующим образом:

$$S(Sx, Sy, Sz, 1) = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

т. е. $[x, y, z, 1] * S(Sx, Sy, Sz) = [Sx * x, Sy * y, Sz * z, 1]$.

Общее изменение масштаба

Общее изменение масштаба получается за счет 4-го диагонального элемента, т. е.

$$[x \ y \ z \ 1] * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & S \end{bmatrix} = [x \ y \ z \ S] = [x^* \ y^* \ z^* \ 1] = \left[\frac{X}{S}, \frac{Y}{S}, \frac{Z}{S}, 1 \right].$$

Такой же результат можно получить при равных коэффициентах частичных изменений масштабов. В этом случае матрица преобразования такова:

$$S = \begin{bmatrix} \frac{1}{S} & 0 & 0 & 0 \\ 0 & \frac{1}{S} & 0 & 0 \\ 0 & 0 & \frac{1}{S} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Трехмерный сдвиг

Недиагональные элементы матрицы 3×3 осуществляют сдвиг в трех измерениях, т. е.

$$[x \ y \ z \ 1] * \begin{bmatrix} 1 & b & c & 0 \\ d & 1 & f & 0 \\ h & i & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x+yd+hz, \ bx+y+iz, \ cx+fy+z, \ 1].$$

Трехмерное вращение

Двухмерный поворот, рассмотренный ранее, является в то же время трехмерным поворотом вокруг оси Z . В трехмерном пространстве поворот вокруг оси Z описывается матрицей

$$R_z(\Theta) = \begin{bmatrix} \cos(\Theta) & \sin(\Theta) & 0 & 0 \\ -\sin(\Theta) & \cos(\Theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Матрица поворота вокруг оси X имеет вид

$$R_x(\Theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\Theta) & \sin(\Theta) & 0 \\ 0 & -\sin(\Theta) & \cos(\Theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Матрица поворота вокруг оси Y имеет вид

$$R_y(\Theta) = \begin{bmatrix} \cos(\Theta) & 0 & -\sin(\Theta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\Theta) & 0 & \cos(\Theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Результатом произвольной последовательности поворотов вокруг осей x, y, z является матрица

$$A = \begin{bmatrix} a & b & c & 0 \\ d & e & f & 0 \\ h & i & j & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Получение проекций

Каждую из проекций можно описать матрицей 4×4 . Этот способ оказывается удобным, поскольку появляется возможность объединить матрицу проецирования с матрицей преобразования.

Получение одноточечной перспективной проекции

Центральная (перспективная) проекция получается путем перспективного преобразования и проецирования на некоторую двумерную плоскость «наблюдения». Центр проекции находится в точке с координатами $(0,0,-k)$ (рис. 10.2), плоскость проецирования $Z = 0$. Перспективная проекция в этом случае обеспечивается преобразованием:

$$[X Y Z H] = [x y z 1]^* \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x y 0 (rz+1)].$$

где $r = \frac{1}{k}$ (рис. 10.2).

Точки на картинной плоскости получаются после данного преобразования путем нормализации

$$\begin{aligned} x^* &= \frac{X}{H} = \frac{x}{rz+1}; \\ y^* &= \frac{Y}{H} = \frac{y}{rz+1}; \\ z^* &= \frac{Z}{H} = \frac{0}{rz+1}, \end{aligned}$$

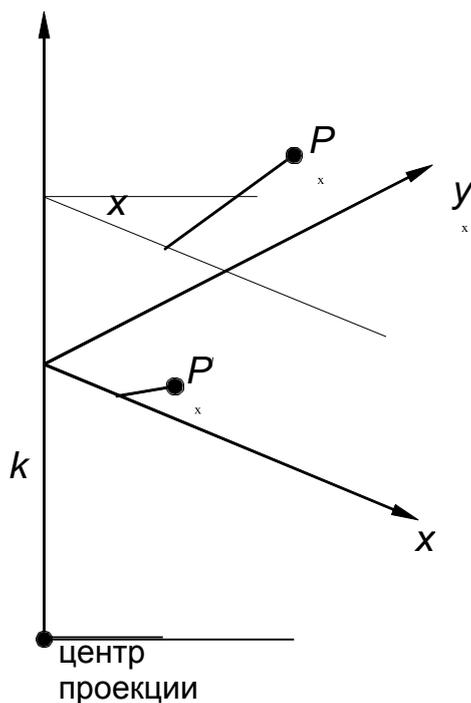


Рис. 10.2. Вычисление одноточечной перспективы

Получение косоугольных проекций

Рассмотрим теперь косоугольную проекцию (рис. 10.3), матрица может быть записана исходя из значений α и l .

Проекцией точки $P(0,0,1)$ является точка $P(l \cos\alpha, l \sin\alpha, 0)$, принадлежащая плоскости xy . Направление проецирования совпадает с отрезком PP , проходящим через две эти точки. Это направление есть $P' - P = (l \cos\alpha, l \sin\alpha, -1)$. Направление проецирования составляет угол β с плоскостью xy .

Теперь рассмотрим проекцию точки x, y, z и определим ее косоугольную проекцию (x_p, y_p) на плоскости xy :

$$\begin{aligned} x_p &= x + z(l \cos\alpha); \\ y_p &= y + z(l \sin\alpha). \end{aligned}$$

Таким образом, матрица 4×4 , которая выполняет эти действия и, следовательно, описывает косоугольную проекцию, имеет вид

$$M_{\text{кос}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ l \cos \alpha & l \sin \alpha & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

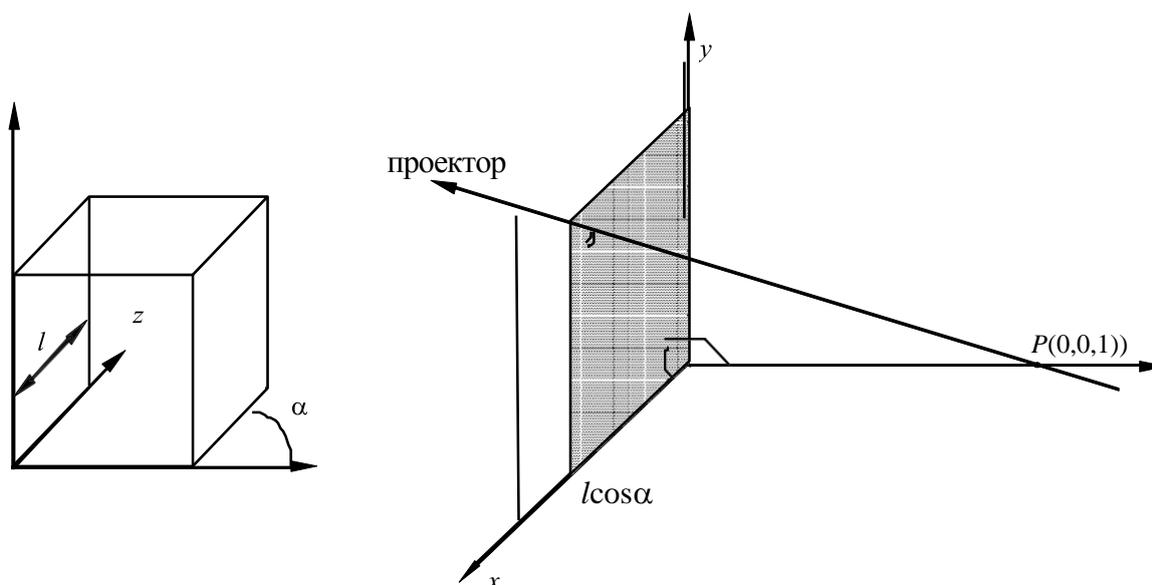


Рис. 10.3. Косоугольные проекции

Применение матрицы $M_{\text{кос}}$ приводит к сдвигу и последующему проецированию объекта: плоскости с постоянной координатой $z = z_1$ переносятся в направлении x на $z_1 l \cos \alpha$ и в направлении y на $z_1 l \sin \alpha$ и затем проецируются на плоскость $z = 0$. Сдвиг сохраняет параллельность прямых, а также углы и расстояния в плоскостях, параллельных оси z .

Для проекции Кавалье $l = 1$, поэтому угол $\beta = 45^\circ$. Для проекции Кабине $l = 1/2$, а $\beta = \arctg(2) = 63,4^\circ$. В случае ортографической проекции $l = 0$ и $\beta = 90^\circ$, поэтому матрица ортографического проецирования является частным случаем косоугольной проекции.

Построение вида спереди

Получение вида спереди возможно также с помощью матрицы 4×4 , которая имеет следующий вид

$$M_{\text{орт}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Как видим, третий столбец матрицы полностью обращает z координату в ноль.

Программная реализация

Программная реализация данной лабораторной весьма похожа на реализацию преобразований на плоскости. Дополнительно описываются матрица для получения проекций и матрица для хранения двухмерного представления.

Вычислительный процесс, запускаемый при нажатии на какую либо кнопку, приведен на рис. 10.4.



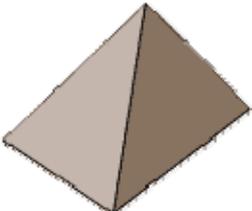
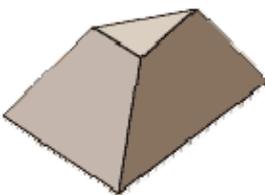
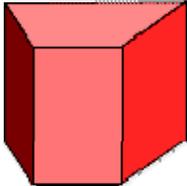
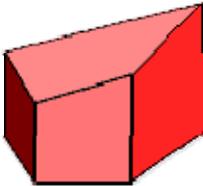
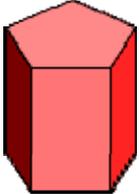
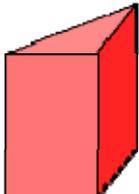
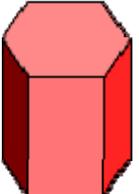
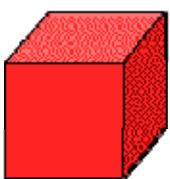
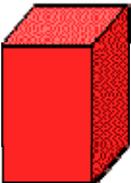
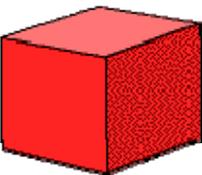
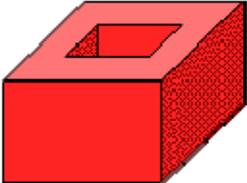
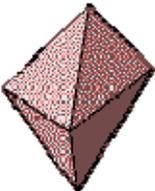
Рис. 10.4. Процесс визуализации трехмерного тела

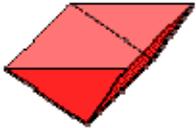
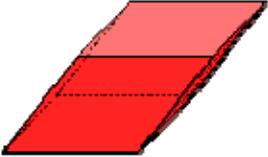
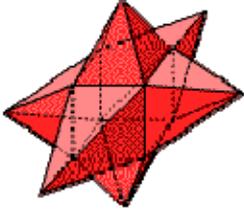
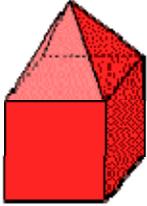
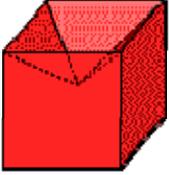
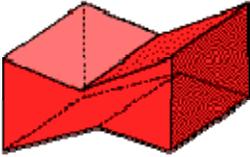
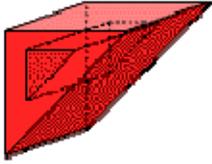
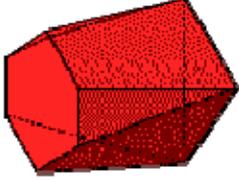
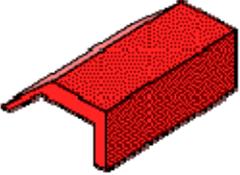
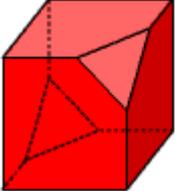
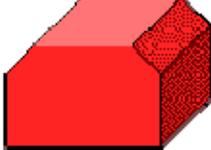
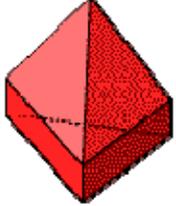
На четвертом этапе, при умножении на матрицу проецирования, исходная матрица тела не должна меняться, поскольку в этом случае будет потеряна информация о координате z . Поэтому результат умножения на этом этапе сохраняется в специальной матрице, по которой и происходит отрисовка.

Задание по лабораторной работе

Реализовать с заданным телом все виды преобразований в пространстве: перенос вдоль координатных осей, отражение относительно основных плоскостей, масштабирование, поворот на заданные углы относительно координатных осей. Предусмотреть восстановление исходной позиции тела. Реализовать получение следующих проекций: вид спереди, центральной одноточечной, косоугольной. Управление организовать как через интерфейсные элементы (меню, кнопки, строки редактирования и пр.), так и через «горячие» клавиши.

Варианты заданий

			
Задание № 1	Задание № 2	Задание № 3	Задание № 4
			
Задание № 5	Задание № 6	Задание № 7	Задание № 8
			
Задание № 9	Задание № 10	Задание № 11	Задание № 12

 <p>Задание № 13</p>	 <p>Задание № 14</p>	 <p>Задание № 15</p>	 <p>Задание № 16</p>
 <p>Задание № 17</p>	 <p>Задание № 18</p>	 <p>Задание № 19</p>	 <p>Задание № 20</p>
 <p>Задание № 21</p>	 <p>Задание № 22</p>	 <p>Задание № 23</p>	 <p>Задание № 24</p>

11. ЛАБОРАТОРНАЯ РАБОТА «ПОСТРОЕНИЕ ТРЕХМЕРНЫХ СЦЕН НА XAML»

Цель лабораторной работы: изучить принципы построения статических 3D сцен с помощью XAML (eXtensible Application Markup Language) в WPF (windows presentation foundation). Написать и протестировать программу для получения трехмерной сцены.

Система координат и размещение камеры

Реализация трехмерных сцен в приложении WPF возможно, как в разметке XAML, так и в процедурном коде, используя предоставляемые платформой .Net возможности двумерной графики. В данной лабораторной работе рассмотрим возможности XAML.

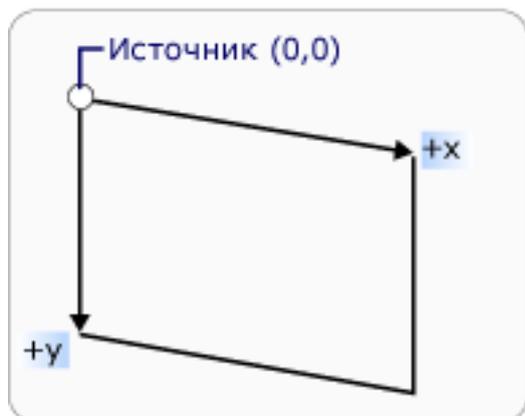
Графическое содержимое трехмерных сцен в приложении WPF инкапсулировано в элементе Viewport3D, который может участвовать в структуре двумерного элемента. Графическая система рассматривает объект Viewport3D как двумерный визуальный элемент в ряду других элементов приложения WPF. Объект Viewport3D функционирует как окно просмотра трехмерной сцены. Говоря точнее, это поверхность, на которую проецируется 3D сцена.

В традиционном приложении двухмерный элемент Viewport3D используется как любой другой контейнерный элемент, например «Grid» или «Canvas».

Начало системы координат WPF для двухмерной графики расположено в левом верхнем угле области отрисовки (обычно экрана или окна). В этом случае положительные значения оси x откладываются вправо, а положительные значения оси y – сверху вниз. Однако в трехмерной системе начало координат располагается в центре отрисовываемой области, положительные значения оси x откладываются вправо, оси y – снизу вверх, а оси z – из центра к наблюдателю (рис. 11.1). WPF принята правосторонняя система координат, поэтому указание положительного значения угла поворота приведет к повороту против часовой стрелки вокруг оси.

Поскольку трехмерная сцена выглядит по-разному в зависимости от точки наблюдения, необходимо указать эту точку наблюдения. Задание точки наблюдения позволяет класс **Camera**. Для описания способа проецирования используются классы **OrthographicCamera** и **PerspectiveCamera** (для получения ортогографической и перспективной проекции соответственно).

2D Система координат



2D Система координат

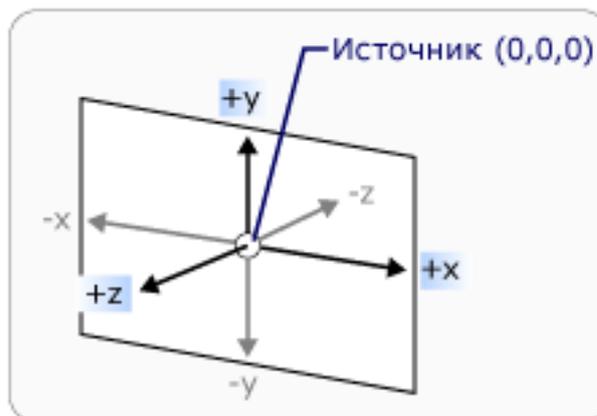


Рис. 11.1. 2D и 3D Системы координат

Свойства **NearPlaneDistance** и **FarPlaneDistance** камеры определяют расстояния до передней секущей плоскости и задней секущей плоскости области видимого объема.

Рассмотрим пример описания получения перспективной проекции⁵:

```
<Viewport3D.Camera>  
  <PerspectiveCamera Position="-  
    250,250,200"  
    LookAtPoint="0,0,0"  
    UpDirection="0,1,0"  
    FieldOfView="40"  
    NearPlaneDistance="1"  
    FarPlaneDistance="500"  
  />
```

Свойство **Position** задает позицию камеры в мировых координатах (центр проекции). Свойство **LookAtPoint** задает вектор, который определяет направление просмотра камеры в мировых координатах. Свойство **UpDirection** задает объект вектор, определяющий направление вверх для камеры. Свойство **FieldOfView** определяет угол между боковыми секущими плоскостями видимого объема.

Освещение сцены

Источники света в трехмерной графике выполняют ту же роль, что и реальные источники света: они делают поверхности видимыми. Более того, источники света определяют, какая часть сцены будет включена в проекцию. Объекты источников света в приложении WPF создают различные эффекты света и тени. Они смоделированы на основе пове-

⁵ При получении перспективной проекции мы определяем центр проекции, следовательно определяем место положения камеры.

дения различных реальных источников света. Сцена должна включать, как минимум, один источник света, иначе модели будут невидимыми.

Указанные ниже источники света являются производными от базового класса **Light**:

AmbientLight: создает рассеянное освещение, при котором все объекты освещены одинаково, независимо от их расположения или ориентации.

DirectionalLight: создает освещение, аналогичное удаленному источнику света. Направленные источники света имеют свойство **Direction**, которое указывается вектор (объект **Vector3D**), но без заданного местоположения.

PointLight создает освещение, как от ближнего источника света. Источники света **PointLights** занимают определенное положение и испускают свет из этого положения. Объекты на сцене освещаются в зависимости от их положения и расстояния относительно источника света. **PointLightBase** предоставляет свойство **Range**, которое определяет расстояние, за пределами которого модели не будут освещены светом. Класс **PointLight** также предоставляет свойства затухания, определяющие интенсивность ослабления источника света в зависимости от расстояния. Можно указать константу, линейную или квадратичную интерполяцию затухания источника света.

Через свойство **Color** устанавливается цвет источника освещения.

Источники света являются объектами **Model3D**, поэтому можно преобразовывать и анимировать свойства источников света, включая положение, цвет, направление и диапазон.

Пример задания направленного источника белого света приведен ниже.

```
<ModelVisual3D>
  <ModelVisual3D.Content>
    <DirectionalLight Color="#FFFFFF" Direction="-1,-1,-1" />
  </ModelVisual3D.Content>
</ModelVisual3D>
```

Задание объектов трехмерной сцены

Для задания объектов на трехмерной сцене используется класс **GeometryModel3D**, в котором можно определить полигональную сетку, состоящую из треугольников, с помощью класса **MeshGeometry3D**. Каждый треугольник в полигональной сетке представляется с помощью индексов в списке вершин. Список вершин определяется свойством **Positions**, а набор примитивов определяется через свойство **TriangleIndices**.

Приведем пример:

```
<GeometryModel3D>
  <GeometryModel3D.Geometry>
    <MeshGeometry3D
      Positions="-1 -1 0 1 -1 0 -1 1 0 1 1 0"
    >
```

```
TriangleIndices="0 1 2 1 3 2" />  
</GeometryModel3D.Geometry>  
</GeometryModel3D>
```

В итоге на сцене представлены два треугольника с двумя общими вершинами (с индексами «0» и «1»). В списке вершин находится четыре вершины.

При построении полигональной сетки важен порядок перечисления вершин для каждого полигона. Для видимых граней этот порядок должен идти по направлению движения часовой стрелки, для не видимых – в другую сторону. Порядок перечисления вершин позволяет вычислить нормали для полигонов, что необходимо для удаления невидимых граней и расчета освещенности. Однако нормали можно определить также вручную через свойство **Normals**.

Применение материалов к модели

Для определения характеристик поверхности модели приложение WPF использует абстрактный класс **Material**. Конкретные подклассы класса **Material** определяют некоторые характеристики внешнего вида поверхности модели, и каждый из них предоставляет свойство **Brush** (кисть), которому можно передать значение **SolidColorBrush**, **TileBrush** или **VisualBrush**.

Класс **DiffuseMaterial** определяет, что кисть будет применена к модели, как если бы она была освещена рассеянным светом. Использование класса **DiffuseMaterial** больше всего напоминает применение кистей непосредственно в моделях двухмерный; поверхности модели не отражают свет, как блестящие поверхности.

Класс **SpecularMaterial** определяет, что кисть будет применена к модели, как если бы поверхность модели была твердой или блестящей, способной отражать блики. Можно установить степень гладкости или «глянца» текстуры, задав значение свойства **SpecularPower**.

Класс **EmissiveMaterial** позволяет указать, что текстура будет применена, как если бы модель излучала свет, эквивалентный цвету кисти. Это не делает модель светящейся; однако это иначе влияет на затенение, чем если бы текстура была создана с помощью класса **DiffuseMaterial** или **SpecularMaterial**.

Для повышения производительности противоположные поверхности объекта **GeometryModel3D** (границы, которые невидимы, поскольку находятся на противоположной стороне модели относительно камеры) удаляются из сцены. Чтобы указать класс **Material** для применения к противоположной поверхности модели, например плоскости, задайте свойство **BackMaterial** модели.

Следующий пример кода демонстрирует применение сплошного цвета и использование полупрозрачности.

```
<GeometryModel3D.Material>  
  <DiffuseMaterial>  
    <DiffuseMaterial.Brush>  
      <SolidColorBrush Color="Cyan" Opacity="0.3" />  
    </DiffuseMaterial.Brush>  
  </DiffuseMaterial>  
</GeometryModel3D.Material>
```

Трёхмерные преобразования

Каждый объект модели имеет свойство **Transform**, с помощью которого можно перемещать модель, изменять ее направление или размер. Для трёхмерных преобразований используются классы **TranslateTransform3D** (преобразование переноса), **ScaleTransform3D** (масштабирование) и **RotateTransform3D** (поворот). WPF также предоставляет класс **MatrixTransform3D**, который позволяет указать те же преобразования в более коротких матричных операциях.

TranslateTransform3D перемещает все точки в **Model3D** в направлении выбранного вектора смещения со свойствами **OffsetX**, **OffsetY** и **OffsetZ**.

ScaleTransform3D изменяет масштаб модели с помощью указанного вектора масштаба относительно центральной точки. По умолчанию **ScaleTransform3D** растягивает или сжимает вершины по отношению к точке отсчета (0,0,0). Однако, если преобразуемая модель строится не от начала координат, то при ее масштабировании от начала координат она будет находиться «не на своем месте». В то же время, если вершины модели умножаются на вектор масштабирования, операция масштабирования приведет и к преобразованию модели, и к ее масштабированию.

Трёхмерную модель можно поворачивать несколькими способами. При обычном преобразовании поворота указывается ось и угол поворота вокруг этой оси. Класс **RotateTransform3D** позволяет определить **Rotation3D** со свойством **Rotation**. Затем для определения преобразования следует указать свойства **AxisAngle** в **Rotation3D**.

Приведем пример поворота вокруг оси *y* на 40 градусов.

```
<GeometryModel3D.Transform>  
  <RotateTransform3D>  
    <RotateTransform3D.Rotation>  
      <AxisAngleRotation3D Axis="0,1,0" Angle="40" />  
    </RotateTransform3D.Rotation>  
  </RotateTransform3D>  
</GeometryModel3D.Transform>
```

Пример описания простой трехмерной сцены

Приведем полный пример описания простой трехмерной сцены, состоящий из источника света и одного повернутого полигона.

```
<Window x:Class="WpfApplication1.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="300" Width="300">
  <Grid>
    <!-- Viewport3D определяет поверхность и область окна куда проецируется
сцена. -->
    <Viewport3D ClipToBounds="True" Width="150" Height="150">

      <!-- Определяем положение и направление камеры. -->
      <Viewport3D.Camera>
        <PerspectiveCamera Position="0,0,2" LookDirection="0,0,-1"
FieldOfView="45" />
      </Viewport3D.Camera>

      <!-- Определяем дочернии объекты для Viewport3D -->
      <Viewport3D.Children>

        <!-- Устанавливаем направления и цвет источника освещения. -->
        <ModelVisual3D>
          <ModelVisual3D.Content>
            <DirectionalLight Color="#FFFFFF" Direction="-1,-1,-1" />
          </ModelVisual3D.Content>
        </ModelVisual3D>

        <ModelVisual3D>
          <ModelVisual3D.Content>
            <GeometryModel3D>

              <!-- Определяем 3D объект -->
              <GeometryModel3D.Geometry>
                <MeshGeometry3D
                  TriangleIndices="0,1,2"
                  Positions="-0.5,-0.5,0.5 0.5,-0.5,0.5
0.5,0.5,0.5">

                </MeshGeometry3D>
              </GeometryModel3D.Geometry>

              <!-- Зададим материал (цвет) объекта -->
              <GeometryModel3D.Material>
                <MaterialGroup>
                  <DiffuseMaterial Brush="Blue" />
                </MaterialGroup>
              </GeometryModel3D.Material>

              <!-- повернем объект на 40 градусов вокруг оси Y -->
              <GeometryModel3D.Transform>
                <RotateTransform3D>
                  <RotateTransform3D.Rotation>
                    <AxisAngleRotation3D Axis="0,1,0"
Angle="40" />
                  </RotateTransform3D.Rotation>
                </RotateTransform3D>
              </GeometryModel3D.Transform>
            </GeometryModel3D>
          </ModelVisual3D.Content>
        </ModelVisual3D>
      </Viewport3D.Children>
    </Viewport3D>
  </Grid>
</Window>
```

```
        </RotateTransform3D.Rotation>
      </RotateTransform3D>
    </GeometryModel3D.Transform>
  </GeometryModel3D>
</ModelVisual3D.Content>
</ModelVisual3D>
</Viewport3D.Children>
</Viewport3D>
</Grid>
</Window>
```

Задание по лабораторной работе

Создать с помощью описания XAML трехмерную сцену, состоящую из нескольких разноцветных объектов. Добавьте к этой сцене несколько источников света. Поместите в отчет изображение этой сцены, полученное при различных трехмерных преобразованиях. Получите разные типы проекций, результат также поместите в отчет. Самостоятельно изучите возможности работы с текстурами. Добавьте в трехмерную сцену объекты с текстурой.

12. ЛАБОРАТОРНАЯ РАБОТА «ТРЕХМЕРНЫЕ ПРЕОБРАЗОВАНИЯ В WPF»

Цель лабораторной работы: изучить принципы построения Динамических 3D сцен с помощью процедурного кода в WPF. Написать и протестировать программу для анимации трехмерной сцены, состоящей из нескольких объектов.

Связь процедурного кода и объектов описанных в XAML

Среди пространств имен XAML можно выделить одно, которое объявляется почти в каждом файле XAML среды выполнения Windows, – это пространство имен для элементов, которые определены в языке XAML. По соглашению собственное пространство имен языка XAML сопоставляется с префиксом «x».

Префикс «x» (собственное пространство имен языка XAML) содержит несколько программных конструкций, которые часто используются в XAML-коде. Приведем только одну: **x:Name**, которая задает имя экземпляра объекта времени выполнения, созданного в исполняемом коде по итогам обработки элемента, который определяет этот объект в XAML-коде. Включение **x:Name** в код XAML можно рассматривать как объявление именованных переменных в коде.

Поэтому для доступа в процедурном коде к трехмерной модели необходимо включить **x:Name** в **ModelVisual3D**:

```
<ModelVisual3D x:Name="MyModel">
```

```
...
```

```
</ModelVisual3D>
```

Описанной **MyModel** можно будет далее оперировать в процедурном коде. Объект **MyModel** принадлежит классу **ModelVisual3D**.

Трехмерные преобразования в процедурном коде

Рассмотрим пример поворота объекта созданного в XAML вокруг оси y. Предварительно подготовим необходимые объекты в обработчике события **Loaded**.

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    myYRotate = new RotateTransform3D();
    myYAxis = new AxisAngleRotation3D();
    myYAxis.Axis = new Vector3D(0, 1, 0);
    myYAxis.Angle = 7;
    myYRotate.Rotation = myYAxis;
}
```

```

        myTransform1 = new Transform3DGroup();
        MyModel.Transform = myTransform1;
    }

```

myYRotate, **myYAxis**, **myTransform1** описываются выше как поля класса **MainWindow**.

Класс **RotateTransform3D** отвечает за преобразование поворота. С помощью класса **AxisAngleRotation3D** описывается ось, вокруг которой будет произведен поворот. Свойство **Axis** описывает направление оси через объект класса **Vector3D**. Свойство **Angle** задает угол поворота в градусах.

Далее создается объект **myTransform1** класса **Transform3DGroup**, который ставится в соответствие полю **Transform** класса нашей 3D-модели. Этот класс удобно использовать для группы последовательных 3D-преобразований. Фактически **Transform3DGroup** представляет собой коллекцию объектов типа **Transform3D**. Отметим, что на данном этапе описанное преобразование поворота и наша 3D-модель пока ни как не связаны.

Далее создадим на форме кнопку для поворота. Обработчик события этой кнопки будет выглядеть следующим образом.

```

private void button1_Click(object sender, RoutedEventArgs e)
{
    myTransform1.Children.Add(myYRotate);
}

```

Таким образом, при нажатии на кнопку в коллекцию преобразований с 3D-моделью будет добавляться очередной поворот. Вторым подходом, сокращающим количество элементов коллекции, является способ более раннего связывания 3D-модели с 3D-преобразованиями. Например:

```

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    myYRotate = new RotateTransform3D();
    myYAxis = new AxisAngleRotation3D();
    myYAxis.Axis = new Vector3D(0, 1, 0);
    myYAxis.Angle = 0;
    myYRotate.Rotation = myYAxis;

    myXRotate = new RotateTransform3D();
    myXAxis = new AxisAngleRotation3D();
    myXAxis.Axis = new Vector3D(1, 0, 0);
    myXAxis.Angle = 0;
    myXRotate.Rotation = myXAxis;

    myTransform1 = new Transform3DGroup();

    MyModel.Transform = myTransform1;
}

```

```

    myTransform1.Children.Add(myYRotate);
    myTransform1.Children.Add(myXRotate);
}

```

При обработке события нажатия на кнопке, в этом случае, достаточно менять параметры созданных преобразований. Например, увеличивать угол поворота.

```

private void button1_Click(object sender, RoutedEventArgs e)
{
    myYAxis.Angle += 7;
    myXAxis.Angle += 7;
}

```

Применение 3D-преобразований к отдельным элементам 3D сцены

Для применения различных 3D-преобразований к разным элементам 3D сцены мы должны описать эти элементы как различные 3D модели, то есть создать различные объекты класса **ModelVisual3D**.

```

<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="303" Width="312" Loaded="Window_Loaded">
    <Grid>
        <!-- Viewport3D определяет поверхность и область окна куда проецируется
сцена. -->
        <Viewport3D ClipToBounds="True" Width="150" Height="150">

            <!-- Определяем положение и направление камеры. -->
            <Viewport3D.Camera>
                <PerspectiveCamera Position="0,0,2" LookDirection="0,0,-1"
FieldOfView="45"/>
            </Viewport3D.Camera>

            <!-- Определяем дочернии объекты для Viewport3D -->
            <Viewport3D.Children>

                <!-- Устанавливаем направления и цвет источника освещения. -->
                <ModelVisual3D>
                    <ModelVisual3D.Content>
                        <DirectionalLight Color="#FFFFFF" Direction="-1,-1,-1" />
                    </ModelVisual3D.Content>
                </ModelVisual3D>

                <!-- Определяем первую 3D модель -->
                <ModelVisual3D x:Name="MyModel">
                    <ModelVisual3D.Content>
                        <GeometryModel3D>

                            <!-- Определяем 3D объект -->
                            <GeometryModel3D.Geometry>
                                <MeshGeometry3D
                                    TriangleIndices="0,1,2"

```

Positions="-0.5,-0.5,0.5 0.5,-0.5,0.5 0.5,0.5,0.5">

```

        </MeshGeometry3D>
    </GeometryModel3D.Geometry>

    <!-- Зададим материал (цвет) объекта -->
    <GeometryModel3D.Material>
        <MaterialGroup>
            <DiffuseMaterial Brush="Blue"/>
        </MaterialGroup>
    </GeometryModel3D.Material>

    <!-- повернем объект на 40 градусов вокруг оси Y -->
    <GeometryModel3D.Transform>
        <RotateTransform3D>
            <RotateTransform3D.Rotation>
                <AxisAngleRotation3D Axis="0,1,0"
Angle="40" />
            </RotateTransform3D.Rotation>
        </RotateTransform3D>
    </GeometryModel3D.Transform>

    </GeometryModel3D>
    </ModelVisual3D.Content>
</ModelVisual3D>

<!-- Определяем вторую 3D модель -->
<ModelVisual3D x:Name="MyModel2">
    <ModelVisual3D.Content>
        <GeometryModel3D>

            <!-- Определяем 3D объект -->
            <GeometryModel3D.Geometry>
                <MeshGeometry3D
                    TriangleIndices="0,1,2"
                    Positions="-0.5,-0.5,0.5 0.5,-0.5,0.5
0.5,0.5,0.5">
                </MeshGeometry3D>
            </GeometryModel3D.Geometry>

            <!-- Зададим материал (цвет) объекта -->
            <GeometryModel3D.Material>
                <MaterialGroup>
                    <DiffuseMaterial Brush="Red"/>
                </MaterialGroup>
            </GeometryModel3D.Material>

            <!-- повернем объект на 90 градусов вокруг оси Z -->
            <GeometryModel3D.Transform>
                <RotateTransform3D>
                    <RotateTransform3D.Rotation>
                        <AxisAngleRotation3D Axis="0,0,1"
Angle="90" />
                    </RotateTransform3D.Rotation>
                </RotateTransform3D>
            </GeometryModel3D.Transform>

        </GeometryModel3D>
    </ModelVisual3D.Content>

```

```

        </ModelVisual3D>
    </Viewport3D.Children>

    </Viewport3D>
    <Button Content="Поворот 1-Y" Height="23" HorizontalAlignment="Left"
Margin="7,226,0,0" Name="button1" VerticalAlignment="Top" Width="87"
Click="button1_Click" />
    <Button Content="Поворот 2-X" Height="23" HorizontalAlignment="Left"
Margin="103,226,0,0" Name="button2" VerticalAlignment="Top" Width="80"
Click="button2_Click" />
    <Button Content="Поворот 1-X" Height="23" HorizontalAlignment="Right"
Margin="0,226,9,0" Name="button3" VerticalAlignment="Top" Width="87"
Click="button3_Click" />
    </Grid>
</Window>

```

В нашем случае каждая модель представлена всего одним полигоном. Результат приведен на рис. 12.1.

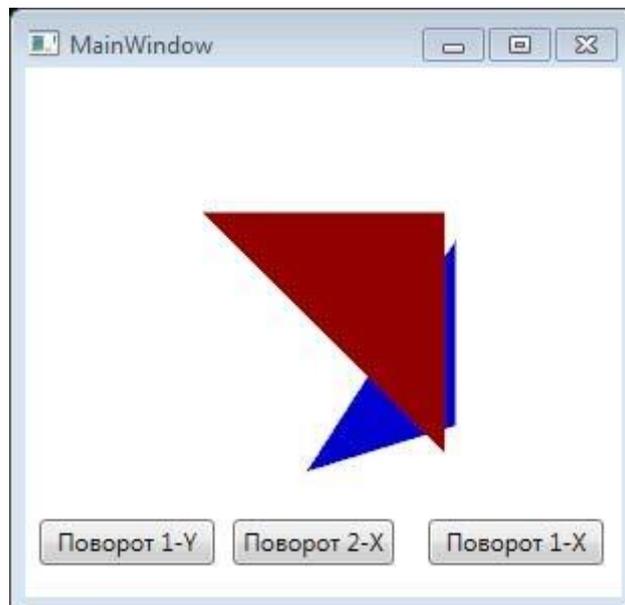


Рис. 12.1. 3D сцена из двух 3D моделей

Далее к каждой 3D модели можно применять свои трехмерные преобразования, как показано ниже.

```

private RotateTransform3D myYRotate, myXRotate;
private AxisAngleRotation3D myYAxis, myXAxis;
private Transform3DGroup myTransform1, myTransform2;

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    myYRotate = new RotateTransform3D();
    myYAxis = new AxisAngleRotation3D();
    myYAxis.Axis = new Vector3D(0, 1, 0);

```

```

myYAxis.Angle = 7;
myYRotate.Rotation = myYAxis;

myXRotate = new RotateTransform3D();
myXAxis = new AxisAngleRotation3D();
myXAxis.Axis = new Vector3D(1, 0, 0);
myXAxis.Angle = 7;
myXRotate.Rotation = myXAxis;

myTransform1 = new Transform3DGroup();
myTransform2 = new Transform3DGroup();

MyModel.Transform = myTransform1;
MyModel2.Transform = myTransform2;
}

private void button1_Click(object sender, RoutedEventArgs e)
{
    myTransform1.Children.Add(myYRotate);
}

private void button2_Click(object sender, RoutedEventArgs e)
{
    myTransform2.Children.Add(myXRotate);
}

private void button3_Click(object sender, RoutedEventArgs e)
{
    myTransform1.Children.Add(myXRotate);
}
}

```

Создание анимации с помощью таймера

В данном разделе рассмотрим создание простейшей анимации без использования специальных классов для анимации, таких как **Vector3DAnimation**, **Vector3DAnimationUsingKeyFrames** и пр. Анимация будет создаваться с помощью таймера.

Для работы с таймером в WPF используется класс **DispatcherTimer**, который инициализируется в процедурном коде следующим образом.

```

MyTimer = new DispatcherTimer();
MyTimer.Tick += new EventHandler(MyTimer_Tick);
MyTimer.Interval = new TimeSpan(100000);

```

Свойство **Interval** определяет частоту срабатывания таймера. Через свойство **Tick** определяется метод (в нашем случае **MyTimer_Tick**), который будет запускаться по тикку таймера. Метод, запускающийся по тикку таймера, например, можно описать следующим образом.

```

private void MyTimer_Tick(object sender, EventArgs e)
{
}

```

Для запуска и остановки таймера необходимо использовать его методы **Start()** и **Stop()**.

Для задания различных трехмерных преобразований к различным объектам не только описываются разные 3D модели, но и создаются различные объекты из класса **Transform3D**. Рассмотрим пример поворота двух граней вокруг осей *y* и *z* в разных направлениях. При описании 3D моделей в отличие от предыдущего примера добавим описание цвета обратной стороны грани. XAML описание для элемента **Viewport3D** будет следующее.

```
<Viewport3D ClipToBounds="True" Width="300" Margin="102,0,102,71">
    <!-- Определяем положение и направление камеры. -->
    <Viewport3D.Camera>
        <PerspectiveCamera Position="0,0,2" LookDirection="0,0,-1"
FieldOfView="45"/>
    </Viewport3D.Camera>

    <!-- Определяем дочерние объекты для Viewport3D -->
    <Viewport3D.Children>

        <!-- Устанавливаем направления и цвет источника освещения. -->
        <ModelVisual3D>
            <ModelVisual3D.Content>
                <DirectionalLight Color="#FFFFFF" Direction="-1,-1,-1" />
            </ModelVisual3D.Content>
        </ModelVisual3D>

        <!-- Определяем первую 3D модель -->
        <ModelVisual3D x:Name="MyModel">
            <ModelVisual3D.Content>
                <GeometryModel3D>

                    <!-- Определяем 3D объект -->
                    <GeometryModel3D.Geometry>
                        <MeshGeometry3D
                            TriangleIndices="0,1,2"
                            Positions="-0.1,-0.1,0.1 0.1,-0.1,0.1
0.1,0.1,0.1">

                            </MeshGeometry3D>
                        </GeometryModel3D.Geometry>

                        <!-- Зададим материал (цвет) объекта -->
                        <GeometryModel3D.Material>
                            <MaterialGroup>
                                <DiffuseMaterial Brush="Blue"/>
                            </MaterialGroup>
                        </GeometryModel3D.Material>

                        <!-- Зададим цвет обратной стороны -->
                        <GeometryModel3D.BackMaterial>
                            <MaterialGroup>
                                <DiffuseMaterial Brush="Green"/>
                            </MaterialGroup>
                        </GeometryModel3D.BackMaterial>
                    </ModelVisual3D.Content>
                </ModelVisual3D>
            </Viewport3D.Children>
        </Viewport3D>
```

```

        <!-- Перенесем объект влево -->
        <GeometryModel3D.Transform>
            <TranslateTransform3D OffsetX="-0.5"/>
        </GeometryModel3D.Transform>

        </GeometryModel3D>
    </ModelVisual3D.Content>
</ModelVisual3D>

<!-- Определяем вторую 3D модель -->
<ModelVisual3D x:Name="MyModel2">
    <ModelVisual3D.Content>
        <GeometryModel3D>

            <!-- Определяем 3D объект -->
            <GeometryModel3D.Geometry>
                <MeshGeometry3D
                    TriangleIndices="0,1,2"
                    Positions="-0.1,-0.1,0.1 0.1,-0.1,0.1
0.1,0.1,0.1">

                </MeshGeometry3D>
            </GeometryModel3D.Geometry>

            <!-- Зададим материал (цвет) объекта -->
            <GeometryModel3D.Material>
                <MaterialGroup>
                    <DiffuseMaterial Brush="Red"/>
                </MaterialGroup>
            </GeometryModel3D.Material>

            <!-- Зададим цвет обратной стороны -->
            <GeometryModel3D.BackMaterial>
                <MaterialGroup>
                    <DiffuseMaterial Brush="Yellow"/>
                </MaterialGroup>
            </GeometryModel3D.BackMaterial>

            <!-- Перенесем объект вправо -->
            <GeometryModel3D.Transform>
                <TranslateTransform3D OffsetX="0.5"/>
            </GeometryModel3D.Transform>

        </GeometryModel3D>
    </ModelVisual3D.Content>
</ModelVisual3D>

</Viewport3D.Children>

</Viewport3D>

```

Обработчик события Loaded будет выглядеть следующим образом.

```

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    //Создаем преобразования для 1 объекта
    myYRotate = new RotateTransform3D();
}

```

```

myYAxis = new AxisAngleRotation3D();
myYAxis.Axis = new Vector3D(0, 1, 0);
myYAxis.Angle = 0;
myYRotate.Rotation = myYAxis;

myZRotate = new RotateTransform3D();
myZAxis = new AxisAngleRotation3D();
myZAxis.Axis = new Vector3D(0, 0, 1);
myZAxis.Angle = 0;
myZRotate.Rotation = myZAxis;

myTransform1 = new Transform3DGroup();
MyModel.Transform = myTransform1;

myTransform1.Children.Add(myYRotate);
myTransform1.Children.Add(myZRotate);

//Создаем преобразования для 2 объекта
myYRotate2 = new RotateTransform3D();
myYAxis2 = new AxisAngleRotation3D();
myYAxis2.Axis = new Vector3D(0, 1, 0);
myYAxis2.Angle = 0;
myYRotate2.Rotation = myYAxis2;

myZRotate2 = new RotateTransform3D();
myZAxis2 = new AxisAngleRotation3D();
myZAxis2.Axis = new Vector3D(0, 0, 1);
myZAxis2.Angle = 0;
myZRotate2.Rotation = myZAxis2;

myTransform2 = new Transform3DGroup();
MyModel2.Transform = myTransform2;

myTransform2.Children.Add(myYRotate2);
myTransform2.Children.Add(myZRotate2);

// Подготавливаем таймер к работа
MyTimer = new DispatcherTimer();
MyTimer.Tick += new EventHandler(MyTimer_Tick);
MyTimer.Interval = new TimeSpan(100000); }

```

Тик таймера выглядит следующим образом.

```

private void MyTimer_Tick(object sender, EventArgs e)
{
    myYAxis.Angle += 1;
    myZAxis.Angle += 1;

    myYAxis2.Angle -= 2;
    myZAxis2.Angle -= 2;
}

```

Остается добавить две кнопки для запуска и остановки таймера и их обработчики.

```

private void button1_Click(object sender, RoutedEventArgs e)
{
    MyTimer.Start();
}

```

```
private void button2_Click(object sender, RoutedEventArgs e)
{
    MyTimer.Stop();
}
```

Задание по лабораторной работе

Разработайте приложение, анимирующее трехмерную сцену в соответствии со своим вариантом.

1. Разработайте программу визуализирующую движение треугольника, прямоугольника, окружности, пирамиды, призмы по разным траекториям с плавно меняющейся скоростью движения.
2. Спроектируйте приложение отображающие различные объекты (пирамидки, кубики, отдельные грани), плавающие в невесомости. Вращение производится также вокруг собственной оси.
3. Реализуйте игру для развития памяти. Перед Вами несколько парных карт. На короткое время игра показывает, где какие карты лежат, затем все карты поворачиваются рубашками вверх. Необходимо вспомнить, где расположены одинаковые карты и выбрать их. При нажатии на карту реализуйте поворот карты вокруг собственной оси. При выборе двух одинаковых карт удалите данные карты по трехмерной траектории.
4. Разработайте программу анимирующую полет бабочки.
5. Создайте приложение анимирующее падение снежинок в 3D пространстве.
6. Реализуйте программу, визуализирующее несколько этапов складывания какого либо оригами.
7. Разработайте приложение анимирующее падение листочков с дерева в 3D пространстве.
8. Спроектируйте приложение отображающее многогранник. При нажатии на кнопку многогранник должен «взорваться», то есть грани должны разлететься в разные стороны, вращаясь вокруг своих осей.
9. Создайте программу визуализации движения планет в солнечной системе по 3D траекториям. Вместо сферической модели планет можно использовать упрощенные тела (куб, пирамид).
10. Разработайте приложение, выводящее 3D модель книги с переворачивающимися страницами.
11. Реализуйте программу, имитирующую полет бумеранга в 3D пространстве.
12. Разработайте 3D сцену пролета по сложным траекториям двух космических кораблей.

13. Создайте приложение, визуализирующее 3D модель удара кием по шару в бильярде.
14. Спроектируйте программу анимирующую бросание игральных костей.
15. Создайте анимированную модель 3D шкафа, с возможностью открывания дверей и выдвижения ящиков.
16. Создайте анимированную модель 3D парусника с косым парусом. Реализуйте возможность движения модели не только в одной плоскости (движение по волнам) и возможность поворота паруса.
17. Разработайте анимированную 3D модель растения со стеблем и листочками.
18. Разработайте анимированную 3D модель цветка.
19. Создайте анимированную 3D модель дома, с возможностью открывания дверей и окон.
20. Спроектируйте приложение для анимации полета двух самолетов (разных фигур высшего пилотажа).