СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

В настоящее время, моделирование реальности с использованием компьютерной графики находится в бурном развитии и приводит к глубоким преобразованиям в жизни человека в разных аспектах: социальном, духовном, экономическом. Эволюция компьютерной графики глубоко повлияла на различные виды масс-медиа, привела к революционным преобразованиям в анимации, кинематографии и в индустрии компьютерных игр. Вдобавок к этому, многочисленные исследования показали, что около 70% из всего своего входного информационного потока, человек поглощает путем визуального восприятия, через изображения, человеческий глаз совместно с визуальным анализатором представляют собой наилучший информационный интегратор. Стала возможной комбинация графики, текста, звука, видеоизображения, анимации для наиболее эффективного представления информации.

Представление дисциплины. Современная *„Компьютерная графика”* является довольно сложной научно-технической дисциплиной, хорошо продуманной и разнообразной. Некоторые из ее разделов, такие как: геометрические преобразования, способы описания кривых и поверхностей, были глубоко изучены, другие продолжают развиваться: методы сканирования растровых изображений, удаление скрытых участков линий и поверхностей, использование цветов и освещение, текстурирование, создание эффектов прозрачностей и бликов и др. Данная работа представляет собой пособие теоретического, методического и прикладного характера. Ее цель состоит в пополнении и закреплении знаний студентов концепциями, теоретическими понятиями и практическими навыками, приобретенными в университетском курсе *„Компьютерная графика”*. В данном курсе рассматриваются не только фундаментальные понятия из области двумерной компьютерной графики, но и технологический уровень, достигнутый в данной области информатики. Широко освещаются понятия, относящиеся к двумерным геометрическим преобразованиям, преобразования визуализации, функции рисования, способы описания кривых и различных форм, техника представления изображений на экране монитора, а также фундаментальные алгоритмы, относящиеся к данной области. Представлены и демонстрационные программы, содержащие подробные комментарии и подтверждающие на практике приобретенные теоретические знания. Все приведенные в данной работе примеры, были реализованы в среде программирования Microsoft Visual C++ и проверены на различных данных. В этом смысле, каждый из представленных примеров, помимо исходного текста программы, содержит и результат выполнения в виде скриншотов с экрана монитора.

Актуальность курса обусловлена необходимостью формирования и развития параметров профессиональной компетенции, необходимой квалифицированным специалистам − программистам, дизайнерам, которые будут работать в различных областях национальной экономики и не только, будучи способными эффективно эксплуатировать существующие графические системы и приложения, а также разрабатывать и внедрять собственные оригинальные программные продукты, посредством использования различных современных специализированных программ и систем.

Место дисциплины в рамках учебной программы. Дисциплина *„Компьютерная графика”* предназначена для студентов III курса, первого цикла университетского образования (Ciclul Licenţă), специальностей *„Информатика”*, *„Прикладная информатика”*, *„Информационный менеджмент”*, *„Прикладная математика”*, будучи разработанной в соответствии с рекомендациями Национального Состава Квалификаций (CNC — Cadru Național al Calificărilor), с учебным планом и распределенной по трем уровням: знания, применения и интегрирования. Является дисциплиной специализации по выбору и включает лекции, лабораторные занятия и индивидуальную работу. Дисциплина включена в учебный план V семестра со 120 часами, из которых 60 являются аудиторными, содержит 4 кредита и *предполагает формирование у студентов специфических компетенций (по специальности), что облегчит консолидацию общих профессиональных компетенций*

Эта дисциплина обеспечивает фундаментальную подготовку, необходимую в изучении других дисциплин по специальности, использующих компьютерную графику. Для успешного освоения этой дисциплины необходимы элементарные познания и начальные практические навыки в дисциплинах родственных компьютерной графике, которые могут быть разделены на две категории:

- дисциплины — технологии, методы и средства которых используются в качестве инструментов в компьютерной графике;
- дисциплины, которые используют компьютерную графику.

Отметим, что дисциплины из обеих этих категорий присутствуют в учебном плане. Из них университетская дисциплина *„Компьютерная графика”* использует понятия, преподаваемые в таких дисциплинах как „Основы программирования”, „Технологии программирования”, „Операционные системы”, „Базы данных”, „Архитектура компьютеров”, „Алгебра”, „Высшая геометрия”, „Дифференциальная геометрия” и др. — все это дисциплины, изучаемые на I и II курсах. Плохое освоение основных понятий, рассматриваемых в данной дисциплине, может привести к затруднениям в изучении и восприятии сложных концепций, вводимых в других дисциплинах специализации.

Основная цель дисциплины заключается в ознакомлении студентов с проблематикой компьютерной графики и вычислительной техники, формирования достижений и компетенций в изучении информационных инструментов, в первую очередь информационных технологий в данной области, что требует знания методов и специальных технологий для представления, запоминания, обработки, передачи и визуализации информации; эти действия должны быть реализованы как в базовом программном обеспечении, так и в прикладном программном обеспечении:

- представление и преобразование объектов;
- графические ядра языков программирования;
- фундаментальные и специализированные алгоритмы компьютерной графики.

Общие цели дисциплины заключаются в профессиональном формировании специалистов IT: предоставлением глубоких теоретических ***знаний*** о: *фундаментальных концепциях, базовых принципах и приемах* из области компьютерной графики; *освоением концепций и технологий* для *проектирования, разработки и использования базового и прикладного программного обеспечения* в

области компьютерной графики; *использованием концепций и технологий* компьютерной графики для *решения задач 2D моделирования и представлений*. ***Теоретико-практические возможности*** использования: *средств, предоставляемых средой программирования* Microsoft Visual C++, с применением библиотеки MFC для проектирования и разработки приложений компьютерной 2D графики; дополнительного *программного обеспечения*: CorelDraw, AdobeIllustrator, CorelPhotopaint, AdobePhotoshop, предназначенного для моделирования и 2D представлений при решении соответствующих задач. ***Способности*** комбинирования и использования знаний графики и возможностей программного обеспечения для компьютерной графики; способности анализировать, объяснять и оценивать корректность выводов в процессе решения задач. В результате лабораторных занятий, студенты должны приобрести ***навыки*** использования самых современных технологий при реализации графических приложений, для использования их в различных областях деятельности человека.

В этом ряду идей, ***профессиональные компетенции***, предлагаемые к развитию в рамках курса *„Компьютерная графика”*, продиктованные необходимостью формирования квалифицированных специалистов в искусстве программирования, как на национальном, так и на международном уровне, могут быть сформулированы следующим образом:

- ***Знание архитектуры компьютеров и операционных систем*** посредством: изучения и описания архитектуры графических систем; использования вычислительных систем в информационных технологиях в области компьютерной графики; применения последовательностей графических преобразований; понимания и оперирования концепциями, теорией и моделями, используемыми в компьютерной 2D графике; проектирования и реализации алгоритмов 2D графики; анализа и оценки графических систем.
- ***Применения компьютерных сетей, системного софта, персональных компьютеров в областях профессиональной деятельности*** посредством: использования в профессиональной деятельности приложений компьютерной графики; внедрения графических библиотек и современных графических 2D редакторов, способствующих использованию приложений компьютерной графики в области профессиональной деятельности.
- ***Использование методов прикладной математики и инструментального софта при решении проблем автоматизации управления производствами*** посредством: применения и комбинирования моделей, информационных и математических инструментов для решения проблем, специфических для компьютерной графики; умения объяснять идеи, задачи и их решение, как специалистам, так и лицам не являющихся специалистами в компьютерной графике; применения новых методов и эффективных алгоритмов для решения задач в области компьютерной 2D графики; развития и внедрения в профессиональную деятельность информационных решений для конкретных задач, используя различные методы компьютерной 2D графики.
- ***Развитие WEB приложений*** посредством: идентификации задач из области WEB, которые могут быть интерпретированы и решены при помощи компьютерной

графики; проведения исследований в области компьютерной 2D графики, с определенным уровнем самостоятельности; критической самооценки, реализации и аргументирования решений в применении к развитию WEB приложений; демонстрации способности построения графической модели сцены объектов, используемых в WEB приложениях; использования самых новых технологий и сред развития программ и информационных систем, специфичных графике, для применения развитых моделей в функциональных продуктах.

- ***Проектирование систем компьютерной графики; систем растровой графики*** посредством: демонстрирования возможности проектирования и разработки графических приложений для решения задач из различных областей деятельности человека; реализации базовых алгоритмов ядра графической системы; разработки графических приложений с использованием языка высокого уровня; сравнительной оценки и аргументирования методов компьютерной графики при разработке конкретных приложений.
- ***Проектирование издательских систем; систем снабженных скриптовыми языками*** посредством: восприятия контекста, в котором для решения задач могут быть использованы методы и приемы компьютерной 2D графики; реализации главных фаз последовательности графических преобразований; проектирования и разработки графических приложений для различных областей использования компьютеров; использования адекватных критериев и методов для оценки информационных продуктов.

Администрирование предмета

Форма обучения	**Код дисципли-ны из учебного плана**	**Ответствен-ный за курс**	**Семестр**	**Часы**					**Оценка**	**Кол-во кредитов**
				Всего	**включая**					
					Ле	**Се**	**Ла**	**ИР**		
Дневная форма обучения	S.05.A.142	**Перетятку С., Кристей М.**	V	120	30	-	30	60	Экза-мен	4

Структура тем предмета

№ п/п	**Составные части**	**Часы**		
		Лекц.	**Лаборат.**	**Индивид. работа**
		дневн.	дневн.	дневн.
1.	Введение в компьютерную графику	2	2	6
2.	Графические 2D библиотеки	6	10	10
3.	Программирование мыши	4	4	10
4.	Графические 2D редакторы	4	10	10
5.	Основные преобразования в плоскости	6	2	10
6.	Представление фигур в однородных координатах	4	0	10

7.	Форматы графических файлов	4	2	4
	Всего	**30**	**30**	**60**

Правила использования пособия. Пособие по дисциплине *„Компьютерная графика”* предоставляет теоретические, методические и прикладные материалы, предназначенные дополнить знания студентов фундаментальными концепциями и практическими примерами программ, разработанных в среде Microsoft Visual C++, представленных как в исходных текстах, так и результатами выполнения в виде скриншотов с экрана монитора. Работа представляет собой методический путеводитель по изучению/преподаванию/оценке указанной дисциплины, в рамках которого определены актуальность, важность, цели, компетенции и завершенность обучения для формирования и развития у студентов профессиональных компетенций в области информационных и коммуникационных технологий. Работа включает содержание дисциплины *„Компьютерная графика”*, в соответствии с куррикулумом последней. Для облегчения получения компетенций, в качестве принципа по организационной структуре содержимого дисциплины был выбран *модульный принцип*. Содержимое было поделено на семь единиц обучения, каждая из которых включала свое содержания, цели изучения, ключевые слова, содержания и учебных ситуаций. Методическое пособие разработано в соответствии установленными содержанием и целями, ориентированными на требования и возможности различных групп студентов. Каждый учебный раздел завершается самооценкой (для демонстрации и анализа уровня усвоения материала). Пособие содержит также список библиографических источников, которые могут служить фундаментом для проведения собственных исследований в области компьютерной графики. В тоже время в работе предлагаются методические рекомендации для индивидуальной работы, которые позволят студентам сделать выбор из предлагаемых работ (небольшие проекты, лабораторные работы, исследования для индивидуальных разработок) и ознакомиться с примерами программ для разработки предусмотренных финальных продуктов. Для заинтересовавшихся проблемами компьютерной графики, предлагаются темы курсовых, дипломных и магистерских работ. На основе предложенных тем и проведении исследований, хотим сориентировать обучающихся на развитие стремления к теоретическим исследованиям, аналитических и прикладных способностей, необходимых для проведения научных исследований в рамках университетского образования по данной дисциплине. Для финальной оценки предлагаются задания, а так же приводятся примеры программ, реализованных в среде программирования Visual C++.

Надеемся, что данная работа будет реально полезна всем увлекающимся и специалистам в таких сложных приложениях графики, анимации и компьютерных игр.

1. . ВВЕДЕНИЕ В КОМПЬЮТЕРНУЮ ГРАФИКУ

Содержание:

1. Предмет изучения компьютерной графикой;
2. Общие концепции компьютерной 2D графики;
3. Области приложения компьютерной графики;
4. Оборудование и программные продукты для графических приложений.

Цели изучения:

- определить понятие компьютерной графики и области ее применения;
- освоить понятия векторной графики, графики ориентированной на пиксели (растровой графики), цветовой модели (RGB, CMY, CMYK, HSB, Lab);
- определять графические устройства (дигитайзеры, сканеры, графические принтеры, графопостроители, мышь, мониторы, видеокарты и др.);
- применять программные продукты для графических приложений (графические библиотеки, специализированные графические программы, графические редакторы, издательские системы – desktop publishing, электронные таблицы – worksheet graphics).

Ключевые слова: компьютерная графика, система, графическое устройство (оборудование), графическое приложение, область приложения, модель, цвет, векторный, растровый, ориентированный на пиксель.

Рассматриваемые вопросы:

1.1. Объект исследования компьютерной графики

Компьютерная графика (англ. *computer graphics*) является разделом информатики, занимающимся генерацией и представлением информации, составляющей окружающую реальность (реалистичные изображения натуральных объектов) в виде визуального контента, спроектированного двумерно на экран монитора или бумажные носители, посредством графических устройств (англ. hardware) и специализированных графических интерфейсов (англ. software). Более того, компьютерная графика позволяет создавать изображения не только объектов реального мира, т.е. натуральных объектов, но и абстрактных, синтетических объектов. Эта характеристика существенно отличает компьютерную графику, как реализатора изображений, от телевидения и фотографии.

В словарях по информатике термин *„Компьютерная графика”* определяется как „*ветвь информатики, отвечающая за автоматическое получение рисунков и изображений*”. Этот термин означает сегодня информационные приложения, связанные с графическими представлениями и обработкой изображений.

Интерактивная компьютерная графики (англ. *interactiv computer graphics*) означает использование в диалоговом режиме графических технологий, в плане понимания и анализа двух- или трехмерных объектов.

Графика, без сомнения, является одной из самых интересных областей использования компьютеров. Когда мы говорим "компьютерная графика", то имеем в виду как двумерную графику для формирования изображений (программы рисования такие, как PaintBrush или Corel), так и автоматизированное проектирование, или программы анимации, или создание изображений для компьютерных игр и т.д.

Мы твердо убеждены, что все те, кто открыл обложку этой работы, имел и не однажды возможность (необходимость) „экспериментировать" с графическим программным обеспечением. Однако далеко не все из читателей осознают, что за спиной этого программного обеспечения находятся множество формул из аналитической геометрии, проективной геометрии, вычислительной геометрии, а также алгоритмы сжатия данных. Ко всему перечисленному добавляется, огромные интеллектуальные усилия и большой объем работы исследователей и программистов.

Подобласти компьютерной графики (согласно ACM - Association for Computing Machinery)

0. *Общие проблемы*.

1. *Архитектура устройств*: устройства вывода; устройства ввода; графические процессоры.

2. *Графические системы*: системы удаленного доступа (англ. remote); независимые системы (англ. stand-alone); распределенные системы.

3. *Генерация рисунков и изображений*: примитивы, символы, объекты; графические представления; геометрические преобразования и вычислительная геометрия; алгоритмы визуализации.

4. *Программы для/или с использованием компьютерной графики*: приложения, которые используют компьютерную графику; графические пакеты; драйверы; языки описания изображений/рисунков; программная поддержка для компьютерной графики.

5. *Моделирование объектов*: кривые и поверхности; иерархия в моделях; пакеты моделирования.

6. *Методы, техники и средства в компьютерной графике*: независимость от устройства; эффективность в выполнении; техника взаимодействия.

7. *Графика 3D и синтез реалистичных изображений*: удаление перекрываемых линий и поверхностей; текстуры, цвета и тени; анимация; развитые алгоритмы.

8. *Разное*: системы видеообщения; компоненты синтеза в мултимедийных приложениях; виртуальная реальность.

Главное содержание дисциплины *„Компьютерная графика"* представляет введение в фундаментальные проблемы этой области информатики и, особенно, изучение, и исследование методов, приёмов и технологий, необходимых для разработки программных продуктов, автоматизирующих процессы ввода, хранения, обработки и вывода графической информации при помощи компьютера.

В качестве дисциплин, тесно связанных с компьютерной графикой, можем перечислить:

1. Цифровая обработка изображений;
2. Проектирование, производство и инженерия с помощью компьютера (CAD/CAM/CAE);
3. Исследование визуального восприятия и распознавание форм;
4. Виртуальная реальность;
5. Технология дисплеев и устройств графического ввода;
6. Взаимодействие человек-компьютер (англ. CHI = Computer-Human Interaction).
7. Мультимедиа.

1.2. Понятие компьютерной 2D графики

Область *Компьютерной графики* требует знания и использования концепций и методов проектирования, разработки и использования базового программного обеспечения и приложений для компьютерной графики. Информационные технологии в этой области требуют знания специальных методов и способов представления, запоминания, обработки, передачи и отображения информации. Эти действия должны реализовываться как в базовом, так и в прикладном программном обеспечении.

Представление графической информации в вычислительных системах, представляет собой совокупность моделей и способов ввода, извлечения и преобразования информации к периферийным устройствам или от них, причем для пользователя эта информация должна иметь именно графический характер.

Под графической системой понимается совокупность оборудования и программ, специализированных на синтезе, обработке и анализе графической информации, представляемой в форме изображений. Пользователь реализует геометрическую модель заданием графической системе команд следующих трех категорий:

- команды, которые идентифицируют или генерируют графические примитивы (полилинии, эллипсы, прямоугольники и т.д.);
- команды, реализующие геометрические преобразования примитивов (масштабирование, поворот, перенос и др.);
- команды, комбинирующие примитивы посредством теоретико-множественных операций (объединение, пересечение, разность).

Основными функциями графической системы являются следующие функции:

- хранения графической информации;
- обработки графической информации;
- чтения/записи графической информации.

Получение изображения на устройстве отображения в графической системе подразумевает два вида процессов:

- процесс формирования изображения (в котором выполняются команды пользователя);
- процесс вывода изображения (в котором реализуется визуализация графической информации).

Запоминание изображений может быть выполнено одним из следующих двух способов:

- *изображения векторного типа*, для которых формат запоминания является векторным (например, отрезок прямой линии запоминается в виде признака отрезка и координат концевых точек);
- *изображения растрового типа (bitmap)*, для которых формат записи является поточечным (кодировка изображения реализуется построчным заданием состояния каждой точки, состоянием, определяемым ее цветом).

Типы графики. Существует множество критериев и подкритериев классификации графических представлений, однако самые важные категории подразделяются на следующие критерии:

- область использования графического представления;
- поведение во времени;
- используемая цветность.

По *области использования* различаются:

1. Традиционная (обычная) графика:
   - технический чертеж: эскиз, план, схема (электрическая, функциональная, блок);
   - диаграммы;
   - карты;
   - картограммы.
2. Художественная графика:
   - художественный рисунок;
   - картина.

По *поведению во времени* графические изображения делятся на следующие виды:

- статические рисунки;
- динамические рисунки (анимация).

В зависимости от *используемого представления цветов* в графических изображениях различают:

- монохроматические изображения (выполненные единственным цветом, отличным от фонового, чаще всего: черно-белые рисунки);
- изображения в оттенках серого цвета;
- цветные изображения (в различных цветовых моделях и палитрах).

Компьютерная графика опирается на концептуальное строение автоматических расчетов, основывается на аксиомах Евклидовой геометрии и применяет инструменты аналитической геометрии с использованием перспективных и ортогональных проекций. Компьютерная графика включает следующие задачи:

- кодирование и манипулирование графическими данными при помощи компьютера;
- математические основы геометрических представлений;
- математическое моделирование атрибутов графических представлений;
- алгоритмы для элементарных графических представлений;
- графические функции в языках высокого уровня;
- алгоритмы для сложных графических представлений.

1.3. Области применения компьютерной графики

Области использования компьютерной графики расширялись одновременно с ростом вычислительных возможностей компьютеров и с удешевлением графических адаптеров и акселераторов: реализация пользовательских интерфейсов с применением многочисленных утилит и сред программирования, автоматизация проектирования (CAD – Computer Aided Design), интерактивные графические представления, автоматизация проектирования в архитектуре и индустриальной графике, визуализация научных и деловых данных, технологии мултимедия, виртуальная и аугментная реальность.

В тоже время, появление многочисленных библиотек, языков и утилит (англ. toolkits) которые предлагают различные способы реализации графических приложений, усложнило программисту выбор наиболее подходящего способа решения поставленной задачи.

Компьютерная графика применяется в различных областях деятельности человека: ядерной физике, электронике, энергетике, топографии и геодезии, картографии и метеорологии, строительстве и архитектуре, медицине и фармакологии, воспитании и обучении, экономике и рекламе, управлении и отслеживании различных процессов. Список может быть продолжен, до перечисления всех областей деятельности в современной жизни, все-таки, в продолжение, выделим несколько самых важных областей использования компьютерной графики, которых, с большой долей вероятности, непосредственно, может коснуться будущая профессиональная деятельность нынешних студентов:

- CAD (Computer Aid Design) – Автоматизированное проектирование (AutoCAD, COMPAS, Qcad);
- CAM (Computer Aid Manufacture) – Автоматизированное производство (ESPRIT, SmartCAM);
- CAE (Computer Aid Engineering) – Автоматизация инженерных расчётов (ANSYS, OpenFOAM);
- CASE (Computer Aid Software Engineering) – Автоматизированная разработка программного обеспечения (Rational Rose, Enterprise Architect);
- OIS (Office Information System) – Офисные информационные системы и мультимедийные системы (Microsoft PowerPoint, Windows Media Player);
- DP (Desktop Publishing) – Цифровое издательское дело (QuarkXPress, Adobe InDesign, Microsoft Publisher, Microsoft Word, Corel Draw, Adobe Illustrator);
- GIS (Geographic Information System) – Геоинформационные системы (Google Earth);
- WD (Web Design) – Веб дизайн;
- CGM (Computer Game Manufacture) – Производство компьютерных игр.
- Виртуальная реальность, Виртуальное оборудование и (Virtual reality, Virtual instrumentation) - LabView, Matlab, Производство фильмов, Виртуальное моделирование в науке и искусстве, E-learning.

1.4. Устройства и программные продукты, используемые в графических приложениях

Устройства и оборудование, предназначенные для использования в графических приложениях, классифицируются по следующим категориям:

- **интерактивные графические устройства** являются логическим устройствами взаимодействия с графической системой:
  - графические терминалы (мониторы, экраны, дисплеи);
  - клавиатура: для ввода строк символов;
  - локатор: идентифицирует позицию и/или направление (джойстик, трекбол, мышь, дигитайзер, сенсорная панель, световое перо);
  - сканер: предназначен для ввода и запоминания уже нарисованного на бумаге изображения в память компьютера;
  - селектор (прерыватель): выбирает указанный вариант из нескольких возможных альтернативных вариантов (функциональные клавиши, кнопки мыши);
  - вариатор: позволяет осуществлять ввод вещественных числовых значений (вращательные потенциометры);
- **пассивные графические устройства** представляют устройства рисования в графических системах:
  - принтеры;
  - графопостроители.

Клавиатура выдает компьютеру последовательность двоичных кодов в соответствии нажатыми пользователем клавишами. Эти последовательности интерпретируются обычным способом как коды символов. С другой стороны, группы клавиш могут интерпретироваться как входная графическая информация. Например, клавиши со стрелками, могут быть использованы для перемещения курсора по экрану.

Джойстик представляет вертикальный рычажок (англ. stick), смонтированный на подставке, что позволяет перемещать курсор в любом направлении. Содержит два потенциометра выходы, которых соответствуют угловой позиции рычажка. Изменение позиции курсора на экране и выбор осуществляются изменением наклона рычажка и нажатием на него.

Трекбол реализует такие же функции, как и джойстик, но отличается от него конструктивно. Вместо рычажка используется небольшой шар, который можно вращать. Вращения воспринимаются четырьмя оптическими датчиками и переводятся в коды перемещения курсора вдоль осей двумерной системы координат. То есть в этом случае перемещения курсора получаются за счет вращения шара вместо наклона рычажка. Как джойстик, так и трекбол были расширены на устройства, позволяющие отслеживать перемещения в пространстве 3D.

Спейсбол (более развит, чем джойстик и трекбол) представляет жесткую сферу, которую пользователь может перемещать в любую позицию, задавая направление или перемещение в пространстве 3D.

Мышь механическая или оптическая. Механическая имеет два взаимно перпендикулярных колесика, которые воспринимают ее перемещения по двум направлениям, x и y, а оптическая основывается на прерывании отражения от поверхности с чередованием отражающих и не отражающих линий (точек). Мышь рассматривает физический экран монитора компьютера, как матрицу точек, представляющих виртуальный экран, используя пары виртуальных координат для локализации соответствующих точки или объекта на экране монитора.

Световой карандаш (англ. light pen) позволяет выбирать позиции на экране посредством восприятия света, получаемого от пикселей на экране CRT. Координаты, получаемые от подобных устройств, используются графическим приложением для выбора или позиционирования объектов на экране.

Дигитайзеры (англ. digitizer) – устройство для ввода в ЭВМ координат точек или кривых линий. Ввод осуществляется обычно с готового изображения (это может быть рисунок, схема, план).

Сканеры (англ. scanner) – устройство предназначено для поточечного ввода в память ЭВМ изображений путём их числового кодирования.

Графические принтеры (англ. printer) – устройство для вывода изображений с экрана монитора на бумагу, плёнку и другие носители. Чаще всего используются струйные и лазерные принтеры.

Плоттеры или графопостроители (англ. plotter) – устройство для автоматического вычерчивания с большой точностью рисунков, схем, сложных чертежей, карт и другой графической информации на бумаге размером до A0 или кальке. Графопостроители рисуют изображения с помощью пера. Используются в системах автоматизированного проектирования. Плоттеры классифицируются по следующим типам:

- рулонные или планшетные;
- перьевые, струйные или электростатические;
- векторные или растровые.

Графические приложения реализуются путем использования следующих классов программных продуктов:

- **графические редакторы** – предназначены для графической обработки в области автоматизированного проектирования (Adobe Ilustrator, Corel Draw, Corel PhotoPaint, Adobe PhotoShop);
- **графические библиотеки -** предназначены для графической обработки посредством языков программирования высокого уровня;
- **специализированные графические программы –** предназначены для решения проблем из различных областей, предоставляя пользователю возможность формулировать задачи как можно проще и в соответствии с используемым в его области языком. В общем случае, эти программы состоят из двух частей: ядра и интерфейса с пользователем (Volfram, Mathematica, Maple, Mathcad)**;**
- **издательские системы** – предназначены для подготовки публикаций (газет, журналов, рекламы и т.д.) в бюро, проходя все этапы классической типографии

(Pagemaker, XPress, SuperPaint, Publish IT, WordStar, WordPerfect, MultiMate, TeX, LaTeX, Xerox Ventura Publisher)**;**

- **приложения для бюротики (worksheet graphics) –** предназначены для приложений учета в финансово-бухгалтерских областях, в которых выходные данные могут быть представлены в виде диаграмм, графиков, таблиц и др. Самыми известными приложениями с такими функциями являются Lotus и Quattro Pro.

Задания для самоконтроля:

1. Опишите основные области применения компьютерной графики.
2. Определите предмет изучения компьютерной графикой.
3. Поясните, в чем заключаются сходство и различие между векторным и растровым представлениями графической информации.
4. Охарактеризуйте в деталях графические устройства: плоттер, дигитайзер, сканер, графический принтер, мышь, монитор.
5. Охарактеризуйте программные продукты для графических приложений: специализированные графические программы, издательские системы, системы для бюротики.
6. Перечислите основные функции графической системы.
7. Аргументируйте методологию исследования компьютерной графики.
8. Оцените роль, смысл и важность компьютерной графики.

2. ГРАФИЧЕСКИЕ 2D БИБЛИОТЕКИ

Содержание:

1. Понятие графической библиотеки;
2. Понятие логического устройства отображения;
3. Создание и использование перьев для рисования различных линий и фигур различными стилями;
4. Основные функции рисования;
5. Создание и использование кистей;
6. Более сложные функции. Рисование фигур (дуга, окружность, эллипс, кривая, полилиния, прямоугольник, многоугольник и др.);
7. Рисование текста;
8. Создание и использование шрифтов.

Цели изучения:

- определить логическое устройство отображения;
- создавать и использовать перья;
- знать основные функции рисования;
- знать более сложные функции рисования;
- создавать и использовать кисти;
- Освоить рисование фигур с заливкой, рисование дуг, хорд, секторов окружностей и эллипсов, рисование полилиний и многоугольников;
- создавать шрифты и применять их при рисовании текстов.

Ключевые слова: текущая позиция, графический курсор, перо, кисть, функция рисования, шрифт.

Рассматриваемые вопросы:

2.1. Понятие графической библиотеки

Под *графическими примитивами* понимаются базовые элементы, которые программисты и пользователи могут использовать для реализации рисунков в том или ином приложении. В качестве графических примитивов выступают точки, отрезки прямых линий, символы, прямоугольники, сплайновые кривые, специальные символы и др.

Графические библиотеки предназначены для графической обработки посредством языков программирования высокого уровня. Эти библиотеки содержат графические процедуры (примитивы), реализуемые графическими функциями необходимыми графическим приложениям.

Как правило, графическая библиотека должна содержать следующие процедуры:

- процедуры инициализации графической сессии, которые выбирают графический режим работы, резервируют и устанавливают области памяти, необходимые для записи и вывода изображений;
- процедуры задания зоны координат видимых точек рисунка (пространство пользователя, область видимости) и активной зоны на экране (область вывода);
- процедуры выбора цветов;
- процедуры для установки атрибутов, таких как цвет линии, тип линии, толщина линии и др.;
- процедуры рисования линий, дуг, эллипсов, полилиний, штриховки внутренности замкнутых фигур и др.;
- процедуры вставки текстовых фрагментов в формируемое изображение;
- процедуры вывода изображений на принтер;
- процедуры управления видеопамятью.

2.2. Понятие логического устройства отображения

Одним из центральных понятий графической библиотеки VC++ является понятие логического устройства отображения. Оно представляет собой поверхность, на которой осуществляется рисование точек, линий, шрифтов, фигур и т.д. Словосочетание логическое устройство (Logic Device) означает, что возможно рисование на любом физическом устройстве отображения: экран, принтер, плоттер или любое другое устройство с отображением двумерной информации. При этом не вдаются в детали, какое именно устройство используется или какова его марка. С точки зрения Windows, Device Context представляет собой структуру, которая описывает (содержит) параметры по умолчанию, используемые в процессе рисования, например, при проведении отрезка прямой линии, используются толщина, длина, цвет.

В отличие от других структур Windows программа пользователя не имеет прямого доступа к полям структуры, соответствующей логическому устройству отображения ЛУО. Она имеет возможность лишь изменять атрибуты посредством набора функций, стандартизирующих доступ.

Device Context стандартизирует все базовые функции для рисования в рамках операционной системы Windows. Весь программный код в операционной системе, который производит какие либо графические операции, осуществляется посредством двух библиотек типа DLL. Первая из этих библиотек, всегда является библиотека GDI.DLL (GDI – Graphics Device Interface) она работает с приложениями, предоставляя в их распоряжение ряд функций стандартного доступа (набор функций рисования), независящих от используемых устройств. Вторая же библиотека зависит от используемого устройства и может быть специфична к монитору, принтеру и т.д. Например, если приложение будет рисовать в клиентской области окна на экране VGA монитора, то будет использоваться библиотека VGA.DLL, если же приложение будет выводить графическую информацию на принтер Epson FX-80, то будет использоваться библиотека Epson9.DLL.

Библиотека базовых классов от фирмы Microsoft (MFC) содержит набор классов, которые упрощают взаимодействие с объектами GDI, находящимися на нижнем уровне. Например, класс CDC обеспечивает работу с устройствами отображения DC. Этот класс содержит богатый набор функций рисования, преобразования координат, операций отсечения для реализации графических приложений. Все остальные классы типа DC являются более специальными, наследуются от класса CDC и расширяют его.

2.3. Создание и использование перьев

Перья представляют собой базовые объекты GDI. Они реализованы в самом ядре операционной системы Windows и находятся там практически с её возникновения. Они используются при проведении линий и кривых. При рисовании замкнутых фигур, перья используются для проведения контура, в то время как кисти отвечают за заливку внутренностей.

Пример создания пера:

```
CPen pen1(PS_SOLID, 3, RGB(255, 0, 0)); // тип линии(следа)
                                        // рисуемой пером,
                                        // толщина линии, цвет линии
```

Для представления цветов в библиотеке определён тип COLORREF, который является целым беззнаковым числом длиной 32 бита. В этих 32 битах упаковывается информация о значениях базовых компонентов Red, Green, Blue, формирующих цвет. Для облегчения составления цветов, в библиотеке есть макроопределение RGB(r,g,b), которое формирует целое число, соответствующее цвету на основе трёх величин r, g, b – представляющих собой значения базовых цветов. Создавая перо можно поступать следующим образом:

```
COLORREF rgbMagenta = RGB(255, 0, 255);
CPen pen2(PS_DASHDOT, 1, rgbMagenta);
```

Типы перьев, определенные в библиотеке, представлены в таблице 2.1.

Таблица 2.1. Типы перьев

Тип пера	Вид рисуемой линии
PS_SOLID	Сплошная (————————————)
PS_DASH	Сегментированная (– – – – – – – – – – – – – –)
PS_DOT	Пунктирная (- - - - - - - - - - - - - - - - - -)
PS_DASHDOT	Сегментнопунктирная (– - – - – - – - – - – - – -)

Выбор перьев. Прежде, чем приступить к рисованию с созданным пером, необходимо его выбрать для использования в рамках логического устройства отображения (Context Dispozitiv). ЛУО имеет слот для пера, в котором в каждый момент времени может находиться только одно перо. Это перо называется *текущим пером*.

При выборе нового пера в качестве текущего, предыдущее перо освобождается.

Для установки нового текущего пера, можно воспользоваться методом `SelectObject()`.

Пример:

```
pDC->SelectObject(&pen1);
```

В качестве параметра указывается адрес нового текущего пера. Как результат, функция `SelectObject()` возвращает адрес предыдущего текущего пера.

Пример:

```
CPen penSolid(PS_SOLID, 5, RGB(0, 200, 0));
CPen *pOldPen = NULL;
pOldPen = pDC->SelectObject(&penSolid);
// ... Рисуем зеленым пером
pDC->SelectObject(pOldPen);
```

Пример: Выбор по очереди двух перьев

```
CPen penGreen(PS_SOLID, 5, RGB(0, 255, 0)); // Зеленое перо
CPen penBlue(PS_SOLID, 3, RGB(0, 0, 255)); // Синее перо
CPen *pOldPen = NULL;
pOldPen = pDC->SelectObject(&penGreen); // Выбираем первое перо
// ... Рисуем зеленым пером
pDC->SelectObject(&penBlue); // Меняем перо
// Функция возвращает адрес зеленого пера, но он не используется
// ... Рисуем синим пером
pDC->SelectObject(pOldPen); // Возвращаемся к изначальному перу
```

ЗАМЕЧАНИЕ. В заключение необходимо восстановить исходное, оригинальное (от операционной системы) перо, чтобы быть уверенным, что Windows не начнет рисовать кнопки и другие контролы вашим пером!

Для рисования не обязательно создавать свое собственное перо. СО Windows предоставляет в распоряжение набор предопределенных перьев, так называемых *сток перьев* (см. таблицу 2.2). Они сконфигурированы со значением атрибутов по умолчанию и обычно используются при рисовании контролов на рабочей поверхности.

Таблица 2.2. Сток предопределенных перьев

Имя сток пера	Описание
`BLACK_PEN`	Тонкая чёрная линия
`WHITE_PEN`	Тонкая белая линия
`NULL_PEN`	Нет линии (рисование текущим фоном)

`NULL_PEN` чаще всего используется при рисовании замкнутых, заливаемых фигур без контура.

Предопределенные перья можно использовать двумя способами:

1. Создаём своё собственное перо на базе сток пера:

```
CPen pen3; // создаём собственное пустое перо

pen3.CreateStockObject(BLACK_PEN); // заполняем своё перо
                                   //  параметрами сток пера
pDC->SelectObject(&pen3); // выбираем своё перо
```

2. Непосредственно выбираем нужное стокперо:

```
pDC->SelectStockObject(NULL_PEN);
```

Созданные перья можно очищать от параметров и загружать в них новые атрибуты.

Например:

```
CPen pen4(PS_DASH, 1, RGB(128, 128, 128));
…
pDC->SelectObject(&pen4);
… // рисование

pen4.DeleteObject();// очищаем перо pen4 от параметров
                    // (удаляется объект GDI из памяти, объект
                // pen4 остается и может быть использован далее)
pen4.CreatePen(PS_SOLID, 4, RGB(128, 128, 128));
… // рисование
```

Стирание перьев. Если уже не нужны атрибуты ранее созданного пера, его можно очистить. В этом случае будет освобождаться объект GDI, находящийся в основе и высвобождаются ресурсы системы. Один и тот же объект перо класса `CPen` можно разгружать и загружать новыми параметрами многократно. Объект GDI, лежащий в основе, высвобождается функцией `DeleteObject()`, членом класса `CPen`.

Пример использования функций `DeleteObject()` и `CreatePen()`

```
CPen pen2 (PS_DASH, 1, RGB(128, 255, 255));
CPen *pOldPen = NULL;
pOldPen = pDC->SelectObject(&pen2);
// Рисуем прерывистым пером
...
pen2.DeleteObject();
pen2.CreatePen(PS_SOLID, 5, RGB(200, 20, 5));
// Рисуем непрерывным пером
...
pDC->SelectObject(pOldPen);}
```

ЗАМЕЧАНИЕ

1. `DeleteObject()` автоматически вызывается, когда завершается время жизни объекта `pen2`.

2. Функция, загружающая перья новыми атрибутами, имеет такие же параметры, как и функция конструктор: `CreatePen(nPenStyle, nWidth, crColor)`, будучи обязанной установить новые стиль, толщину и цвет линии.

2.4. Основные функции рисования

При рисовании используется понятие текущего положения. Для непосредственной установки текущего положения используется функция `MoveTo()`, которая имеет две формы:

```
CPoint CDC::MoveTo(int x, int y)
CPoint CDC::MoveTo(CPoint point)
```

Эта функция устанавливает текущее положение в точку с координатами `(x, y)` или `point`. В качестве результата функция возвращает объект точку, представляющий собой старое текущее положение.

Например:

```
CPoint cp;
cp = pDC->MoveTo(100,75);
CPoint p(100,75);
pDC->MoveTo(p);
```

Функция `LineTo()` рисует отрезок текущим пером от текущего положения до указанной точки, в которую и перейдёт новое значение текущего положения. У функции также две формы:

```
BOOL CDC::LineTo(int x, int y)
BOOL CDC::LineTo(CPoint point)
```

Текущее положение устанавливается в точку с координатами (x, y) или point. Функция возвращает ненулевое значение, если рисование прошло успешно и нулевое значение в противном случае (например, не готово устройство отображения). Попала или не попала линия в границы рабочей области неважно.

Например, следующий фрагмент нарисует пунктирный треугольник при помощи функции `LineTo()`.

```
CPen penRed(PS_DOT, 1, RGB(255, 0, 0)); // Создаем перо
CPen*pOldPen = NULL; // Объявляем указатель для запоминания
                     // адреса оригинального пера (пера от ОС)
pOldPen = pDC->SelectObject(&penRed); // Выбираем красное перо
pDC->MoveTo(50, 100);
pDC->LineTo(100, 100); // Рисуем основание
pDC->LineTo(75, 50);   // Рисуем первую боковую линию
pDC->LineTo(50, 100);  // Рисуем вторую боковую линию
pDC->SelectObject(pOldPen); // Восстанавливаем оригинальное перо
```

Результат работы данного фрагмента, представлен на рис. 2.1.

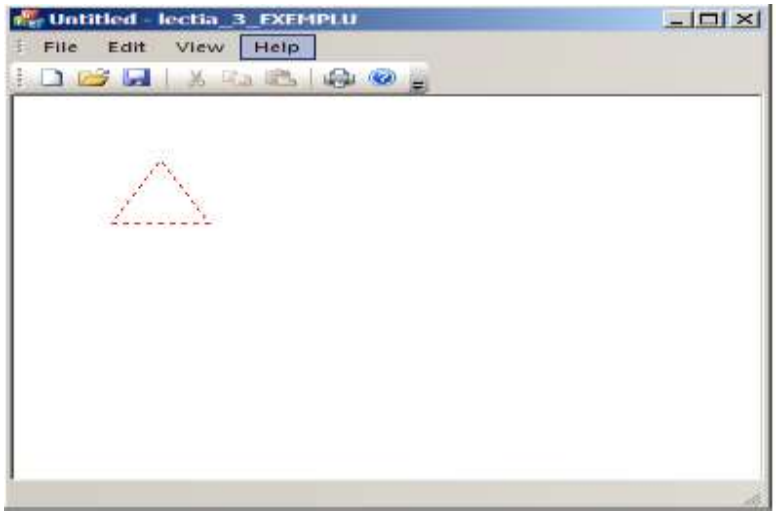

Рис. 2.1. Использование пера со стилем `PS_DOT`

Функция

```
CPoint CDC::GetCurrentPosition()
```

возвращает объект точку `CPoint`, представляющую текущее положение (ту, куда привел последний вызов `MoveTo()` или `LineTo()`).

Функция

```
COLORREF CDC::SetPixel(int x, int y, COLORREF crColor)
```

или

```
COLORREF CDC::SetPixel(POINT point, COLORREF crColor)
```

рисует пиксель в позиции `(x, y)` или `point` цвета `crColor`. Функция возвращает предыдущий цвет пикселя в этой точке, в случае, когда указанная позиция попадает внутрь рабочей области приложения, и возвращает -1, если указанная позиция не попадает в эту область.

При помощи функции

```
COLORREF CDC::GetPixel(int x, int y) const
```

или

```
COLORREF CDC::GetPixel(POINT point) const
```

можем узнать цвет пикселя в указанной позиции. Функция вернет значение -1, если указанная позиция не попадает в рабочую область приложения.

Практически в любой графической программе возникает необходимость в информации о рабочей области приложения. Запрос такой информации осуществляется так:

```
CRect rcClient;
GetClientRect(&rcClient); // Узнаем прямоугольник, окаймляющий
                          // рабочую область окна приложения
```

Еще несколько полезных функций:

```
rcClient.CenterPoint() // Возвращает объект CPoint, совпадающий
                       // с центром прямоугольника
```

```
rcClient.Width()  // Ширина прямоугольника
rcClient.Height() // Высота прямоугольника
```

Далее, рассмотрим демонстрационный пример для лабораторной работы № 1 **„Графические библиотеки. Базовые функции”.**

Постановка задачи:

1. Даны функция $y=f(x)$ и её разложение в числовой ряд, например, $y=\cos(x)$. Составить на языке C++ собственную функцию (процедуру), зависящую от двух параметров и вычисляющую значение функции f для заданного значения аргумента x с указанной точностью ε.
2. Создать проект приложение типа Single document в Microsoft Visual C++. Добавить в него код, рисующий отрезками прямых линий график функции f, указанным цветом на заданном интервале (a, b) с заданным шагом h. Для

вычисления значений функции использовать собственную процедуру и указанную точность.

3. Добавить в проект код, рисующий другим цветом поверх первого графика поточечный график этой же функции, но вычисляющий её в процессе рисования при помощи стандартной библиотеки математических функции. Графики должны совпасть.
4. Добавить в проект код, создающий два текстовых файла с одинаковым количеством строк, совпадающим с количеством вычисляемых точек графика. Первый файл должен содержать для каждой точки: значение аргумента; значение функции, вычисленное при помощи собственной процедуры; значение функции, вычисленное при помощи библиотечных функции; абсолютную величину разности двух вычисленных значений функции; заданное значение точности. Этот файл использовать для визуального контроля того факта, что абсолютная величина разности в каждой строке не превосходит значение точности. Второй файл должен содержать для каждой строчки только значение аргумента и значение функции, вычисленное при помощи собственной процедуры. Числа в строках должны разделяться запятой или пробелом. Второй текстовый файл использовать в следующем пункте для дополнительного контроля.
5. Запустить специализированную программу построения графиков функций. Импортировать второй текстовый файл и построить график функции с использованием пар координат из текстового файла. Построить поверх полученного графика график этой же функции путём её аналитического задания. Графики должны совпасть. Провести анализ полученных результатов.

Пример:

Рассмотрим пример программы, строящей график функции с использованием графической библиотеки в Visual C++. После того как будет создан проект Single Document с именем “Grafic” среди классов этого проекта будет и класс `CGraficView`. В этом классе будет функция **`void CGraficView::OnDraw(CDC* pDC)`**, которая отвечает за перерисовку содержимого рабочей области окна нашего приложения. Она будет вызываться всякий раз, когда с окном происходят изменения, например, пользователь мышью уменьшает или увеличивает размеры окна. В эту функцию мы и будем добавлять код, рисующий график функции cos. Перед заголовком функции `OnDraw()` подключим библиотеку математических функции и запишем прототип нашей функции – процедуры, вычисляющей значения косинуса.

```
// CGraficView drawing
#include "math.h"

double mycos(double x, double eps)
{
 ...
}
```

В качестве параметра, функция `OnDraw()` получает указатель на объект, представляющий собой текущее логическое устройство отображения (в нашем случае экран монитора). Это объект класса `CDC` (Class Device Context), отвечающего за работу с устройствами отображения и содержит большое количество функции вывода графической информации. Ими мы и воспользуемся при рисовании графика. В саму функцию `OnDraw()` после комментария

`// TODO: add draw code for native data here`

добавляем код.

```
void CGraficView::OnDraw(CDC* pDC)
{
 CGraficDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 if(!pDoc)
  return;

 // TODO: add draw code for native data here
 double a, b;

 // Определяем интервал табулирования
 a = -2.0 * 4.0 * atan(1.0);
 b = 2.0 * 4.0 * atan(1.0);

// a = -30.0 * 4.0 * atan(1.0);
// b =  30.0 * 4.0 * atan(1.0);

 // Определяем точность вычислений
 // double precision = 0.0000000000000001;
 double precision = 0.0001;

 // Создаём цвета для перьев
 COLORREF MYCOSCOLOR = RGB(255, 0, 0); // Красный
 COLORREF STCOSCOLOR = RGB(0, 255, 0); // Зеленый

 // Создаём перья для рисования
 CPen penMyCos(PS_SOLID, 1, MYCOSCOLOR);
 CPen penAxis(PS_SOLID, 1, RGB(0, 0, 255));
 CPen* pOldPen = NULL;

 // Определяем размеры зоны рисования
 CRect rcClient;
 GetClientRect(&rcClient);

 // Выбираем перо и рисуем оси
 pOldPen = pDC->SelectObject(&penAxis);

 // Рисуем горизонтальную ось
 pDC->MoveTo(0, rcClient.CenterPoint().y);
 pDC->LineTo(rcClient.Width(), rcClient.CenterPoint().y);

 // Рисуем вертикальную ось
```

```
 pDC->MoveTo(rcClient.CenterPoint().x, 0);
 pDC->LineTo(rcClient.CenterPoint().x,
             rcClient.Height() - 1);

 double scalex = 60.0;
 //double scalex = 100.0;
 double scaley = 100.0;
 unsigned int graphsegments = 200;
 //unsigned int graphsegments = rcClient.Width();
 double x, y, step;

//Меняем перо и рисуем график нашей функции отрезками прямых
//линий
 pDC->SelectObject(&penMyCos);

 x = a;
 step = (b - a) / graphsegments;
 y = mycos(x, precision);
 pDC->MoveTo(rcClient.CenterPoint().x + (int)(x*scalex),
             rcClient.CenterPoint().y - (int)(y*scaley));
for(unsigned int i = 1; i <= graphsegments; i++)
 {
  x += step;
  y = mycos(x, precision);
  pDC->LineTo(rcClient.CenterPoint().x + (int)(x * scalex),
              rcClient.CenterPoint().y - (int)(y * scaley));
 }
 // Подпись для нашего графика
 pDC->SetTextAlign(TA_TOP+TA_RIGHT);
 pDC->SetTextColor(MYCOSCOLOR);
 pDC->TextOut(rcClient.Width() - 1, 0, "y=mycos(x)");

//Чертим поточечно график стандартной функции
 //step = 0.001;
 step = 0.1;
 x = a;
 while(x <= b)
 {
  y = cos(x);
  pDC->SetPixel(rcClient.CenterPoint().x + (int)(x*scalex) + 0,
                rcClient.CenterPoint().y - (int)(y*scaley) + 0,
                STCOSCOLOR);
  pDC->SetPixel(rcClient.CenterPoint().x + (int)(x*scalex) - 1,
                rcClient.CenterPoint().y - (int)(y*scaley) + 0,
                STCOSCOLOR);
  pDC->SetPixel(rcClient.CenterPoint().x + (int)(x*scalex) + 1,
                rcClient.CenterPoint().y - (int)(y*scaley) + 0,
                STCOSCOLOR);
  pDC->SetPixel(rcClient.CenterPoint().x + (int)(x*scalex) + 0,
                rcClient.CenterPoint().y - (int)(y*scaley) - 1,
                STCOSCOLOR);
  pDC->SetPixel(rcClient.CenterPoint().x + (int)(x*scalex) + 0,
                rcClient.CenterPoint().y - (int)(y*scaley) + 1,
```

```
                STCOSCOLOR);
  x += step;
 }

 // Надпись для стандартного графика
 pDC->SetTextAlign(TA_BOTTOM + TA_RIGHT);
 pDC->SetTextColor(STCOSCOLOR);
 pDC->TextOut(rcClient.Width() - 1, rcClient.Height() - 1,
              "y=cos(x)");

 // Надпись для стандартного графика
 pDC->SelectObject(pOldPen);
}
```

Несколько замечаний по поводу реализации своих функций. Для начала запишем несколько элементов разложения функции в ряд:

Подметим закономерность, что, зная очередное слагаемое, следующее получается из предыдущего, изменением знака, домножением на x^2 и делением на $(2k\text{-}1)$ и $2k$. Отметим, что *cos* функция чётная, т.е. $cos(-x)=cos(x)$. Будем считать, что требуемая точность достигается, когда очередное слагаемое станет по абсолютной величине меньше, чем *eps*. Отметим ещё, что все вычисления будем вести с плавающей точкой двойной точности (`double`). Первый вариант собственной функции выглядит следующим образом:

```
double mycos(double x, double eps)
{
 double x2,S,p,k;

 if(x<0.0)
  x=-x;
 x2=x*x;
 S=p=1.0;
 k=0.0;

 while(fabs(p)>eps)
 {
  p=-p*x2;
  k+=1.0;
  p/=k;
  k+=1.0;
  p/=k;
  S+=p;
 }
 return S;
}
```

Данная функция будет хорошо работать только для небольших значений *x*. Для улучшения функции необходимо провести дополнительный анализ. В нашем случае воспользуемся тем, что *cos* функция периодичная. Поэтому значение аргумента *x* можно свести к интервалу $[0,2\pi]$; $cos(x+2\pi)=cos(x)$, а, воспользовавшись и другими свойствами *cos* можно свести значение аргумента к интервалу $[0, \pi/4]$. Доработанный вариант функции mycos выглядит следующим образом:

```
double mycos(double x, double eps)
{
 double x2, S, p, k;
 const double pi    = 3.14159265358979323846264338327950;
 const double dvapi = pi*2.0;
 const double pi_2  = pi/2.0;
 const double pi_4  = pi/4;
 int signum=0;

 if(x<0.0)
  x=-x;
 //Если x>=2pi тогда cos(x)=cos(x-2pi)
 while(x>=dvapi)
  x=x-dvapi;
 //Если x>pi тогда cos(x)=cos(2pi-x)
 if(x>pi)
  x=dvapi-x;
 //Если x>pi/2 тогда cos(x)=-cos(pi-x)
 if(x>pi_2)
 {
  x=pi-x;
  signum=1;
 }

 x2=x*x;
 S=p=1.0;
 k=0.0;

 while(fabs(p)>eps)
 {
  p=-p*x2;
  k+=1.0;
  p/=k;
  k+=1.0;
  p/=k;
  S+=p;
 }

 if(signum)
  S=-S;
 return S;
}
```

Упражнение: Записать окончательный вариант функции mycos(), в котором значение аргумента приводиться к интервалу [0,π/4].

Пояснение: Воспользоваться свойством, если $x \in (\pi/4; \pi/2]$, тогда *cos(x)=sin(pi/2-x).*

```
double mycos(double x, double eps)
{
 const double pi    = 3.14159265358979323846264338327950;
 const double pi_4  = pi/4;
 const double doipi = pi*2.0;
 const double pi_2  = pi/2.0;
 double x2, s, p, pp, k;
 int signum=0;
```

```
 if(x<0.)
  x=-x;
 //Если x>=2pi тогда cos(x)=cos(x-2pi)
 while(x>=doipi)
  x=x-doipi;
 //Если x>pi тогда cos(x)=cos(2pi-x)
 if(x>pi)
  x=doipi-x;
 //Если x>pi/2 тогда cos(x)=-cos(pi-x)
 if(x>pi_2)
 {
  x=pi-x;
  signum=1;
 }
 //Если x>pi/4 тогда cos(x)=sin(pi/2-x)
 if(x>pi_4)
 {
  x=pi_2-x;
  s=p=x;
  k=1.;
 }
 else
 {
  s=p=1.0;
  k=0.;
 }

 pp=p;
 x2=x*x;
 while(pp>eps)
 {
  p=-p*x2;
  k+=1.0;
  p/=k;
  k+=1.0;
  p/=k;
  s+=(pp=p);
  if(pp<0.0)
   pp=-pp;
 }

 if(signum)
  s=-s;

 return s;
}
```

Результат рисования графиков функций представлен на рис. 2.2.

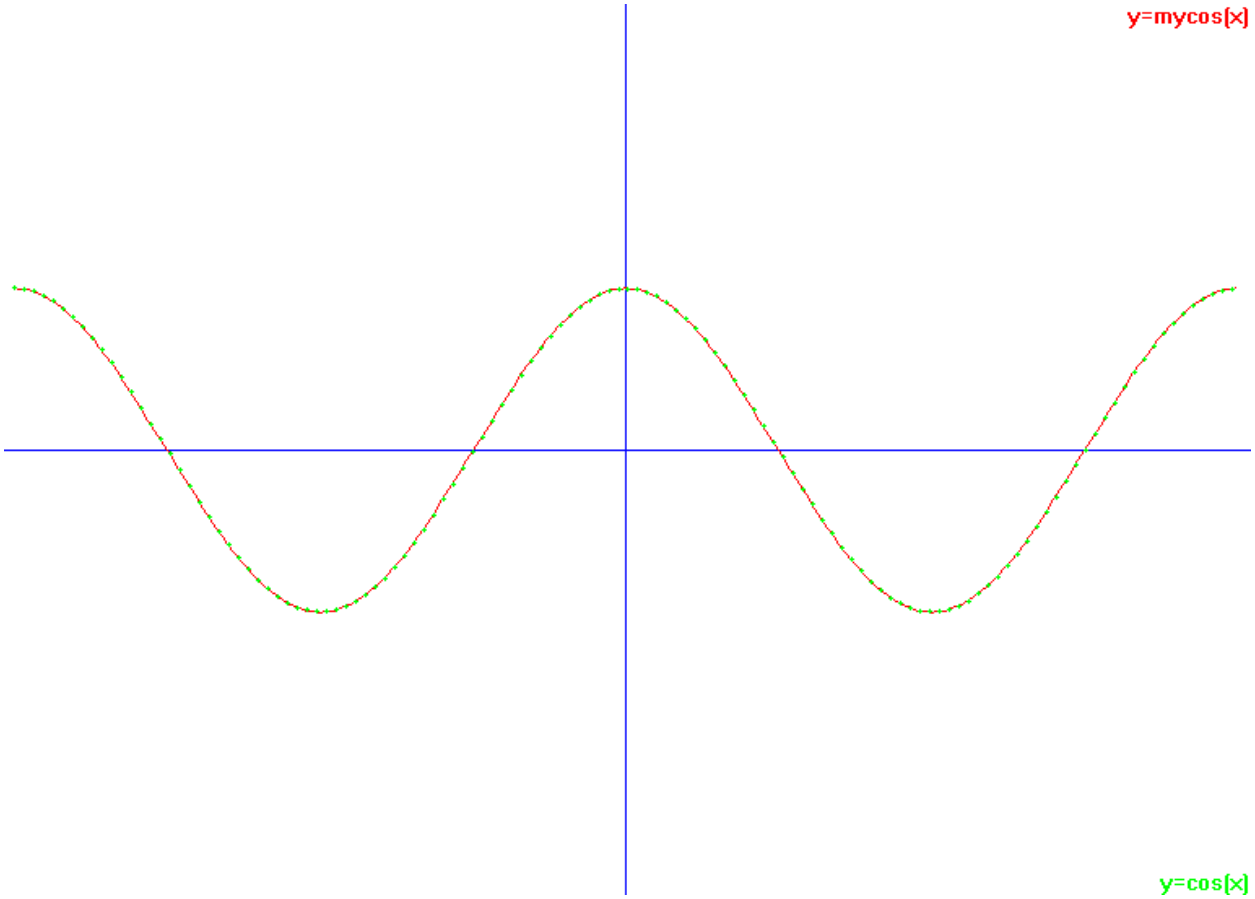

Рис. 2.2. Рисование графика функции

2.5. Создание и использование кистей

Большинство функций рисования используют как перо, так и кисть. Перо используется для проведения контура, а кисть для заполнения внутренности.

Работу с кистями обеспечивает класс `CBrush`. Можно создать кисть со сплошной заливкой или со штриховкой, или основанной на изображении из растрового файла.

При создании кисти можно воспользоваться любым из системных цветов (цвета которые используются операционной системой при оформлении интерфейса).

Создание кистей со сплошной заливкой и со штриховкой.

Проще всего создать кисть со сплошной заливкой:

Достаточно создать объект `CBrush` и передать конструктору единственный параметр – ссылку на желаемый цвет заливки. Таким образом, будет создана кисть, которая будет закрашивать внутренности замкнутых объектов соответствующим цветом, например:

```
CBrush brYellow(RGB(192, 192, 0));
```

Можем закрашивать со штриховкой, указав при создании кисти в качестве первого параметра – индикатор типа штриховки и желаемого цвета – в качестве второго