МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра автоматизации обработки информации

Утверждаю:
Зав. каф. АОИ
профессор
_____Ю.П. Ехлаков
«__» _____2016 г.

Методические указания для выполнения
практических работ по дисциплине

КОМПЬЮТЕРНАЯ ГРАФИКА

для студентов направления подготовки
09.03.04 «Программная инженерия»

Разработчик:
доцент каф. АОИ
_____ Т.О. Перемитина

Томск – 2016

СОДЕРЖАНИЕ

Практическая работа №1 «Знакомство с GIMP»

Цель работы:

- Изучить структурные компоненты окна программы GIMP.
- Освоить процесс подготовки окна программы к работе с изображением (отображать палитры, максимизировать рабочую область).
- Освоить принцип управления масштабом отображения изображения с помощью палитры навигации, с помощью мыши и управляющих кнопок.
- Изучить способы выполнения различных команд в программе GIMP.
- Изучить принцип создания нового документа и задания его параметров (размер, разрешение, цветовую модель).
- Отработать горячие клавиши для наиболее часто выполняемых команд.
- Изучить назначение и способ отображения палитр: navigator, info, color, swatches, styles, history, layers, channels.
- Изучить назначение и способ отображения панелей: options, tools.
- Изучить принцип работы с панелью инструментов GIMP.
- Изучить назначение основных команд меню.
- Изучить принцип выбора конкретного инструмента и задания его параметров на панели options.
- Изучить параметры и принцип воздействия на изображение инструментов выделения, перемещения и кадрирования.
- Изучить параметры и принцип воздействия на изображение инструментов рисования, удаления, заливки.
- Изучить способы заливок и их типы.
- Изучить назначение и принцип создания контуров.
- Изучить принципы работы с контурами (преобразование контура в выделение, заливка и обводка контура, преобразование выделения в контур, сохранение контуров).
- Отработать использование инструментов выделения и перемещения.
- Отработать использование инструментов рисования, удаления, заливки.

Теоретический материал

Существует достаточно много программ, предназначенных для работы с растровыми изображениями. Одной из таких программ является графический редактор GIMP.

GIMP — многоплатформенное программное обеспечение для редактирования изображений (GIMP — GNU Image Manipulation Program).

Редактор GIMP пригоден для решения множества задач по изменению изображений, включая ретушь фотографий, объединение и создание изображений. Программа GIMP многофункциональна. Ее можно использовать как простой графический редактор, как профессиональное приложение по ретуши фотографий, как сетевую систему пакетной обработки изображений, как программу для рендеринга изображений, как преобразователь форматов изображения и т.д. GIMP спроектирован расширяемым, т.е. при помощи дополненийон способен реализовывать любые возможные функции. Передовой интерфейс для разработки сценариев позволяет легко автоматизировать выполнение любых задач разного уровня сложности. 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 WindowsTM или Mac OS XTM 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. Одни характеризуют GIMP как доступный в Linux аналог Photoshop, другие настаивают на том, что принципиально невозможно сравнивать эти две программы, и отмечают, что их интерфейс и основная концепция значительно различаются, а совпадает, строго говоря, только тип обрабатываемых данных – растровые изображения. Это, конечно, не совсем верно: редакторы сходны как минимум еще и тем, что оба принадлежат к «тяжелой весовой категории». В общем, забегая вперед, можно сказать, что наборы встроенных инструментов в них тоже достаточно похожи, и тому, кто знает Photoshop, будет несложно начать работу в GIMP. Но в освоении более сложных средств опыт использования Photoshop не поможет: гибкие и предоставляющие массу возможностей подключаемые модули GIMP организованы совершенно по-другому.

Задание на выполнение

Работа выполняется согласно выданному варианту задания в начале занятия. В работе необходимо использовать:

1. Трансформацию объектов;
2. Работу с цветом и градиентом;
3. Работу с фильтрами изображений.

Практическая работа № 2 «Создание простейшего трехмерного изображения»

Цель работы: Получить навыки моделирования трехмерных объектов.

В OpenGL используются как основные три системы координат: левосторонняя, правосторонняя и оконная. Первые две системы являются трехмерными и отличаются друг от друга направлением оси z: в правосторонней она направлена на наблюдателя, а в левосторонней - в глубь экрана. Для задания различных преобразований объектов сцены в OpenGL используются операции над матрицами, при этом различают три типа матриц: *видовая, проекций* и *текстуры*. Все они имеют размер 4×4. Видовая матрица определяет преобразования объекта в мировых координатах, такие как параллельный перенос, изменение масштаба и поворот. Матрица проекций задает, как будут проецироваться трехмерные объекты на плоскость экрана (в оконные координаты). Для того, чтобы выбрать, какую матрицу надо изменить, используется команда **glMatrixMode**(mode: GLenum), вызов которой со значением параметра mode равным **GL_MODELVIEW**, **GL_PROJECTION**, **GL_TEXTURE** включает режим работы с видовой, проекций и матрицей текстуры соответственно. Для вызова команд, задающих матрицы того или иного типа необходимо сначала установить соответствующий режим.

Для определения элементов матрицы текущего типа вызывается команда **glLoadMatrix**[f d](m: ^GLtype), где m указывает на массив из 16 элементов типа float или double в соответствии с названием команды, при этом сначала в нем должен быть записан первый столбец матрицы, затем второй, третий и четвертый.

Команда **glLoadIdentity** заменяет текущую матрицу на единичную. Для умножения текущей матрицы слева на другую матрицу используется команда **glMultMatrix**[f d](m: ^GLtype), где m должен задавать матрицу размером 4×4 в виде массива с описанным расположением данных. Однако обычно для изменения матрицы того или иного типа удобно использовать специальные команды, которые по значениям своих параметров создают нужную матрицу и перемножают ее с текущей.

В целом, для отображения трехмерных объектов сцены в окно приложения используется следующая последовательность действий:

Координаты объекта => Видовые координаты =>
Усеченные координаты => Нормализованные координаты =>
Оконные координаты

Рассмотрим каждое из этих преобразований отдельно.

Видовое преобразование

К видовым преобразованиям будем относить перенос, поворот и изменение масштаба вдоль координатных осей. Для проведения этих операций достаточно умножить на соответствующую матрицу каждую вершину объекта и получить измененные координаты этой вершины:

$(x^*, y^*, z^*, 1)T = M * (x, y, z, 1)T,$

где M матрица видового преобразования.

Кроме изменения положения самого объекта иногда бывает нужно изменить положение точки наблюдения, что однако также приводит к изменению видовой матрицы. Это можно сделать с помощью команды

gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz: GLdouble)

где точка (eyex,eyey,eyez) определяет точку наблюдения, (centerx, centery, centerz) задает центр сцены, который будет проектироваться в центр области вывода, а вектор (upx,upy,upz) задает положительное направление оси y, определяя поворот камеры.

Проекции

В OpenGL существуют ортографическая (параллельная) и перспективная проекция. Первый тип проекции может быть задан командами

glOrtho(left, right, bottom, top, near, far: GLdouble)

gluOrtho2D(left, right, bottom, top: GLdouble)

Первая команда создает матрицу проекции в усеченный объем видимости (параллелограмм видимости) в левосторонней системе координат. Параметры команды задают точки (left, bottom, -near) и (right, top, -near), которые отвечают левому нижнему и правому верхнему углам окна вывода. Параметры near и far задают расстояние до ближней и дальней плоскостей отсечения по дальности от точки (0,0,0) и могут быть отрицательными.

Во второй команде, в отличие от первой, значения near и far устанавливаются равными -1 и 1 соответственно.

Перспективная проекция определяется командой

gluPerspective(angley, aspect, znear, zfar: GLdouble),

которая задает усеченный конус видимости в левосторонней системе координат. Параметр angley определяет угол видимости в градусах по оси y и должен находиться в диапазоне от 0^0 до 180^0. Угол видимости вдоль оси x задается параметром aspect, который обычно задается как отношение сторон области вывода. Параметры zfar и znear задают расстояние от наблюдателя до плоскостей отсечения по глубине и должны быть положительными. Чем больше отношение zfar/znear, тем хуже в буфере глубины будут различаться расположенные рядом поверхности, так как по умолчанию в него будет записываться «сжатая» глубина в диапазоне от 0 до 1.

Область вывода

После применения матрицы проекций на вход следующего преобразования подаются так называемые усеченные (clip) координаты, для которых значения всех компонент (xc, yc, zc, wc)T находятся в отрезке [-1,1]. После этого находятся нормализованные координаты вершин по формуле:

(xn, yn, zn)T=(xc/wc, yc/wc, zc/wc)T

Область вывода представляет из себя прямоугольник в оконной системе координат, размеры которого задаются командой:

glViewPort(x, y, width, height: GLint)

Значения всех параметров задаются в пикселах и определяют ширину и высоту области вывода с координатами левого нижнего угла (x,y) в оконной системе координат. Размеры оконной системы координат определяются текущими размерами окна приложения, точка (0,0) находится в левом нижнем углу окна.

Используя параметры команды **glViewPort**(), вычисляются оконные координаты центра области вывода (ox, oy) по формулам ox=x+width/2, oy=y+height/2.

Пусть px=width, py=height, тогда можно найти оконные координаты каждой вершины:

(xw, yw, zw)T = ((px/2) xn+ ox , (py/2) yn+ oy , [(f-n)/2] zn+(n+f)/2)T

При этом целые положительные величины n и f задают минимальную и максимальную глубину точки в окне и по умолчанию равны 0 и 1 соответственно. Глубина каждой точки записывается в специальный буфер глубины (z-буфер), который используется для удаления невидимых линий и поверхностей.

Для отработки буфера глубины (проще говоря, для корректного отображения трехмерных объектов и сцен) данную возможность необходимо инициализировать, вызвав команду **glEnable**(**GL_DEPTH_TEST**).

Кроме задания самих примитивов можно определить метод их отображения на экране, где под примитивами в данном случае понимаются многоугольники. Под гранью понимается одна из сторон многоугольника, и по умолчанию лицевой считается та сторона, вершины которой обходятся против часовой стрелки. Направление обхода вершин лицевых сторон можно изменить вызовом команды **glFrontFace**(mode: GLenum) со значением параметра mode равным **GL_CW**, а отменить- с **GL_CCW**.

Чтобы изменить метод отображения многоугольника используется команда **glPolygonMode**(face, mode :GLenum)

Параметр mode определяет, как будут отображаться многоугольники, а параметр face устанавливает тип многоугольников, к которым будет применяться эта команда и может принимать следующие значения:

- **GL_FRONT** для лицевых граней
- **GL_BACK** для обратных граней
- **GL_FRONT_AND_BACK** для всех граней

Параметр mode может быть равен:

- **GL_POINT** при таком режиме будут отображаться только вершины многоугольников.
- **GL_LINE** при таком режиме многоугольник будет представляться набором отрезков.
- **GL_FILL** при таком режиме многоугольники будут закрашиваться текущим цветом с учетом освещения и этот режим установлен по умолчанию.

Кроме того, можно указывать, какой тип граней отображать на экране. Для этого сначала надо установить соответствующий режим вызовом команды **glEnable**(**GL_CULL_FACE**), а затем выбрать тип отображаемых граней с помощью команды **glCullFace**(mode: GLenum)

Вызов с параметром **GL_FRONT** приводит к удалению из изображения всех лицевых граней, а с параметром **GL_BACK** - обратных (установка по умолчанию).

Задание на выполнение:

Согласно варианту задания построить трехмерную сцену с использованием двумерных примитивов OpenGl. Вращать объекты по таймеру.

Практическая работа №3 «Примитивы библиотек GLU и GLUT»

Цель работы: Получить навыки работы с примитивами графических библиотек GLU и GLUT.

Для вывода примитивов из библиотеки GLU необходимо создать указатель на quadric - объект с помощью команды **gluNewQuadric**, затем вызвать одну из команд **gluSphere**(), **gluCylinder**(), **gluDisk**(), **gluPartialDisk**() и, по окончанию использования объекта, вызвать процедуру **gluDeleteQuadric**(). К примеру, код программы можно содержать следующие строки:

```
procedure FormCreate(Sender : TObject)
begin
  …  … { инициализация }
  quadConus := gluNewQuadric;
  gluQuadricDrawStyle(quadConus, GLU_FILL);  // Стиль визуализации
  glNewList (1, GL_COMPILE);
  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @ColorConus);
  glTranslatef(0.0 , -0.4, 0.0);
  gluCylinder (quadConus, 0.25, 0.0, 0.8, 20, 20);
  glEndList;
end;

procedure FormDestroy(Sender : TObject);
begin
  gluDeleteQuadric(quadConus);
  …
  …
end;
```

Рассмотренные строки создадут список под номером 1, содержащий в себе конус (цилиндр с одним нулевым радиусом) с определенными параметрами материала. Особого внимания заслуживает строка **gluQuadricDrawStyle**(qobj, style), определяющая стиль отображения объекта. Параметр style может принимать следующие значения:

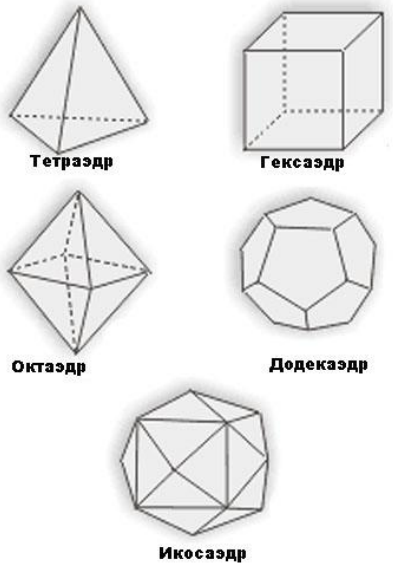
- **GLU_FILL** – отображение цельных, примитивов;
- **GLU_LINE** – примитивы отображаются набором линий;
- **GLU_SILHOUETTE** – примитивы отображаются как силуэт;
- **GLU_POINT** – примитивы состоят из точек.

Рассмотрим команды отдельно: **gluSphere**(qobj: ^GLUquadricObj, radius: GLdouble, slices, stacks: GLint) –строит сферу с центром в начале

координат и радиусом radius. При этом число разбиений сферы вокруг оси z задается параметром slices, а вдоль оси z параметром stacks.

gluCylinder(qobj: ^GLUquadricObj, baseRadius, topRadius, height: GLdouble, slices, stacks: GLint) – строит цилиндр без оснований (то есть кольцо), продольная ось параллельна оси z, заднее основание имеет радиус baseRadius , и расположено в плоскости z =0, переднее основание имеет радиус topRadius и расположено в плоскости z =height. Если задать один из радиусов равным нулю, то будет построен конус. Параметры slices и stacks имеют тот же смысл, что и в предыдущей команде.

gluDisk(qobj: ^GLUquadricObj, innerRadius, outerRadius: GLdouble, slices, loops: GLint) – строит плоский диск с центром в начале координат и радиусом outerRadius. При этом если значение innerRadius ненулевое, то в центре диска будет находиться отверстие радиусом innerRadius. Параметр slices задает число разбиений диска вокруг оси z, а параметр loops - число концентрических колец, перпендикулярных оси z.



gluPartialDisk(qobj: ^GLUquadricObj, innerRadius, outerRadius: GLdouble, slices, loops: GLint, startAngle, sweepAngle: GLdouble) - отличие этой команды от предыдущей заключается в том, что она строит сектор круга, начальный и конечный углы которого отсчитываются против часовой стрелки от положительного направления оси y и задаются параметрами startAngle и sweepAngle.

Команды, проводящие построение примитивов из библиотеки GLUT, реализованы через стандартные примитивы OpenGL и GLU. Для построения нужного примитива достаточно произвести вызов соответствующей команды.

glutSolidSphere(radius: GLdouble, slices, stacks: GLint)

glutWireSphere(radius: GLdouble, slices, stacks: GLint)

Команда **glutSolidSphere**() строит сферу, а **glutWireSphere**() - каркас сферы радиусом radius. Остальные параметры имеют тот же смысл, что и в предыдущих командах.

glutSolidCube(size: GLdouble)

glutWireCube(size: GLdouble)

Эти команды строят куб или каркас куба с центром в начале координат и длиной ребра size.

glutSolidCone(base, height: GLdouble, slices, stacks: GLint)

glutWireCone(base, height: GLdouble, slices, stacks: GLint)

Эти команды строят конус или его каркас высотой height и радиусом основания base, расположенный вдоль оси z. Основание находится в плоскости $z=0$. Остальные параметры имеют тот же смысл, что и в предыдущих командах.

glutSolidTorus(innerRadius, outerRadius: GLdouble, nsides, rings: GLint)

glutWireTorus(innerRadius, outerRadius: GLdouble, nsides, rings: GLint)

Эти команды строят тор или его каркас в плоскости z=0. Внутренний и внешний радиусы задаются параметрами innerRadius, outerRadius. Параметр nsides задает число сторон в кольцах, составляющих ортогональное сечение тора, а rings- число радиальных разбиений тора.

glutSolidTetrahedron / glutWireTetrahedron

Эти команды строят тетраэдр или его каркас, при этом радиус описанной сферы вокруг него равен 1.

glutSolidOctahedron / glutWireOctahedron

Эти команды строят октаэдр или его каркас, радиус описанной вокруг него сферы равен 1.

glutSolidDodecahedron / glutWireDodecahedron

Эти команды строят додекаэдр или его каркас, радиус описанной вокруг него сферы равен квадратному корню из трех.

glutSolidIcosahedron / glutWireIcosahedron

Эти команды строят икосаэдр или его каркас, радиус описанной вокруг него сферы равен 1.

Задание на выполнение:

Согласно варианту задания построить трехмерную сцену, содержащую примитивы из библиотек GLU и GLUT.

Рекомендуемая литература

1. Перемитина Т.О. Компьютерная графика: учеб. пособие. - Томск, ТУСУР, 2012. - 144 с. [Электронный ресурс]: науч.-образовательный портал ТУСУРа. — URL: https://edu.tusur.ru/training/publications/5613

2. Немцова Т.И., Назарова Ю. В. Компьютерная графика и WEB-дизайн. - М.: ФОРУМ, 2013. - 288 с.

3. Перемитина Т.О. Компьютерная графика: методические указания по выполнению лабораторных работ для студентов, обучающихся по направлению подготовки «Программная инженерия». – Томск: ТУСУР, каф. АОИ, 2012. – 23 с. [Электронный ресурс]: науч.-образовательный портал ТУСУРа. — URL: https://edu.tusur.ru/training/publications/5610

4. Перемитина Т.О. Компьютерная графика: метод. рекомендации по выполнению самост. работы для студентов спец-ти 231000.62 «Программная инженерия». – Томск: ТУСУР, каф. АОИ, 2012. – 10 с. [Электронный ресурс]: науч.-образ. портал ТУСУРа. — URL: https://edu.tusur.ru/training/publications/5612