МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Государственное образовательное учреждение
высшего профессионального образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра автоматизации обработки информации

Утверждаю:
Зав. каф. АОИ
профессор
_____Ю.П. Ехлаков
«__» _____2011 г.

Методические указания для выполнения
лабораторных работ
по дисциплине

КОМПЬЮТЕРНАЯ ГРАФИКА

для студентов специальности
230102 - «Автоматизированные системы обработки информации и управления»

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

Томск – 2011

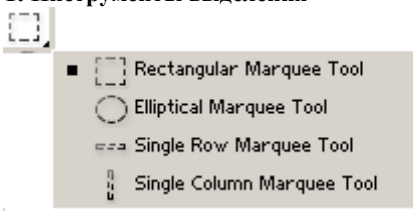
СОДЕРЖАНИЕ

Лабораторная работа № 1 «Знакомство с Photoshop»

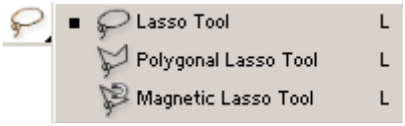
Цель работы: Изучить основные возможности графического редактора Photoshop.

Основные инструменты редактора:

1. Инструменты выделения



- Прямоугольное выделение;
- Овальное выделение;
- Выделение строки пикселов;
- Выделение столбца пикселов;

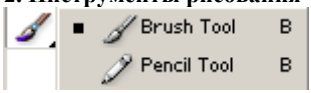


- Лассо – инструмент создания выделенной области в режиме свободного рисования;
- Многоугольное лассо – обрисовка контура области небольшими отрезками;

Магнитное лассо - используется при обрисовке границы между областями, существенно отличающихся по цвету.

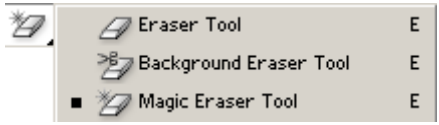
- Волшебная палочка. Позволяет автоматически выделить участки изображения, смежные с указанным по цвету.

2. Инструменты рисования



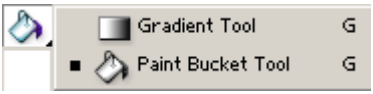
- Кисть;
- Карандаш

.



- Ластик;
- Фоновый ластик;
- Волшебный ластик.

3. Инструменты заливки

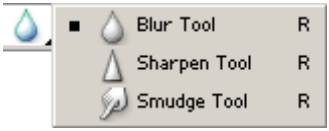


- Заливка.
- Градиент.

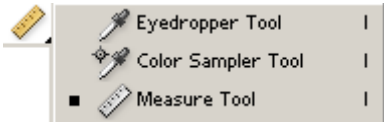
4. Инструменты ретуши

- Штамп – позволяет наложить на любой фрагмент изображения другой его фрагмент;
- Текстура – для его работы используется прямоугольный фрагмент.

- Размытие – уменьшает резкость
- Резкость – увеличивает резкость;
- Палец – «размазывает» фрагмент изображения, создавая плавные, нерезкие переходы цвета



5. Измерительные инструменты



- Пипетка – позволяет получить информацию о цвете произвольной точки изображения;
- Проба цвета – позволяет измерить цвет сразу в нескольких точках;
- Измерение – эквивалент линейки, измеряет расстояния и углы наклона.

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.

Лабораторная работа №2 «Построение и аффинные преобразования двумерных объектов»

Цель работы: Получить навыки моделирования двумерных объектов и применения к ним аффинных преобразований.

Как известно, все изменения изображений можно выполнить с помощью трех базовых операций: ***смещения*** (переноса, перемещения); *масштабирования* (увеличения или уменьшения размеров); поворота изображения (употребляют также термины вращение, изменение ориентации).

Двумерные фигуры представляются в виде трехмерной матрицы с использованием однородных координат, для того чтобы применить следующие аффинные преобразования:

I. Матрица вращения (rotation):

$$[R] = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

II. Матрица растяжения-сжатия:

$$[D] = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \delta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

III. Матрица отражения:

$$[M] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

IV. Матрица переноса (translation):

$$[T] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \lambda & \mu & 1 \end{bmatrix}.$$

Преобразования производятся умножением матриц преобразований на матрицу вершин фигуры и присваиванием новых значений последним. Таким образом, преобразования выполняются над множеством вершин фигуры, после чего результат преобразований отображается с новыми координатами.

Пример: Построить матрицу растяжения с коэффициентом растяжения α вдоль оси абсцисс и β вдоль оси ординат и с центром в точке *A(a,b)*.

Шаг 1. Перенос на вектор *–A(–a,–b)* для смещения центра растяжения с началом координат:

$$[T_{-A}] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -a & -b & 1 \end{bmatrix}.$$

Шаг 2. Растяжение вдоль координатных осей с коэффициентами α и δ.
Матрица преобразования имеет вид:

$$[D] = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \delta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Шаг 3. Перенос на вектор *A(a,b)* для возвращения центра растяжения в прежнее положение:

$$[T_A] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{bmatrix}.$$

Перемножив матрицы в том же порядке $[T_{-A}] \cdot [D] \cdot [T_A]$, получим вид нашего преобразования:

$$(x', y', 1) = (x, y, 1) \cdot \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \delta & 0 \\ a \cdot (1-\alpha) & b \cdot (1-\delta) & 1 \end{bmatrix}.$$

Для выполнения преобразований необходимо знать основные положения матричной алгебры.

Пусть матрица A имеет размерность $k \times m$, матрица B размерности $m \times n$. Результирующая матрица будет иметь порядок $k \times n$:

$$c_{ij} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj}.$$

Важно: умножение матриц не коммутативно: $A \cdot B \neq B \cdot A$.

Задание на выполнение:

Работа выполняется согласно выданному варианту задания.

1. Построить двумерное изображение заданной фигуры.
2. Выполнить масштабирование, сдвиг, поворот фигуры.
3. Выполнить композицию преобразований.

Лабораторная работа №3 «Подключение графической библиотеки OpenGL»

Цель работы: Получить навыки моделирования двумерных объектов OpenGL.

Для построения самой минимальной программы OpenGL требуется выполнить ряд обязательных действий, для понимания которых необходимо знать одно из основных понятий операционной системы Windows – ссылка и контекст устройства. Известно, что ссылка на контекст устройства - величина типа HDC, для получения которой используется функция **GetDC**. Ссылка на контекст устройства содержит характеристики устройства и средства отображения. Прежде чем получить контекст воспроизведения, сервер OpenGL должен получить детальные характеристики используемого оборудования. Эти характеристики хранятся в специальной структуре, тип которой - **TPixelFormatDescriptor** (описание формата пикселя). Структура **PixelFormatDescriptor** - детальное описание графической системы, на которой происходит работа. Формат пикселя определяет конфигурацию буфера цвета и вспомогательных буферов.

```
procedure SetDCPixelFormat (hdc : HDC);
var
 pfd : TPixelFormatDescriptor;
 nPixelFormat : Integer;
begin
 FillChar (pfd, SizeOf (pfd), 0);
      pfd.dwFlags := PFD_DRAW_TO_WINDOW or
PFD_SUPPORT_OPENGL or PFD_DOUBLEBUFFER;
 nPixelFormat := ChoosePixelFormat (hdc, @pfd);
 SetPixelFormat (hdc, nPixelFormat, @pfd);
end;
```

Полям структуры присваиваются желаемые значения, затем вызовом функции **ChoosePixelFormat** осуществляется запрос системе, поддерживается ли на данном рабочем месте выбранный формат пикселя, и вызовом функции **SetPixelFormat** устанавливаем формат пикселя в контексте устройства. Функция **ChoosePixelFormat** возвращает индекс формата пикселя, который нам нужен в качестве аргумента функции **SetPixelFormat**. Заполнив поля структуры **TPixelFormatDescriptor**, мы определяемся со своими пожеланиями к графической системе, на которой будет происходить работа приложения, машина OpenGL подбирает наиболее подходящий к нашим пожеланиям формат, и устанавливает уже его в качестве формата пикселя для последующей работы. Наши пожелания корректируются применительно к реальным характеристикам системы.

Поля структуры "битовые флаги" - dwFlags существенно влияют на работу приложения, нужно учитывать, что некоторые флаги совместно работать не могут, а некоторые могут присутствовать только в паре с другими. Значение флагов:

PFD_DRAW_TO_WINDOW or **PFD_SUPPORT_OPENGL**, означает то, что будет осуществляться вывод в окно, и что система в поддерживает OpenGL.

PFD_DOUBLEBUFFER включает режим двойной буферизации, когда вывод осуществляется не на экран, а в память, затем содержимое буфера выводится на экран. Это очень полезный режим, если в любом примере на анимацию убрать режим двойной буферизации и все команды, связанные с этим режимом, хорошо будет видно мерцание при выводе кадра.

PFD_GENERIC_ACCELERATED имеет смысл устанавливать в случае, если компьютер оснащен графическим акселератором.

Флаги, заканчивающиеся на "**DONTCARE**" , сообщают системе, что соответствующий режим может иметь оба значения, то есть **PFD_DOUBLE_BUFFER_DONTCARE** - запрашиваемый формат пикселя может иметь оба режима - одинарной и двойной буферизации.

Примитивы OpenGl определяются набором из одной или более вершин (vertex). Под вершиной понимается точка в трехмерном пространстве, координаты которой можно задавать следующим образом:

glVertex[2 3 4][s i f d](cords: type)

glVertex[2 3 4][s i f d]v(cords: ^type)

Координаты точки задаются максимум четырьмя значениями: x, y, z, w, при этом можно указывать два (x,y) или три (x,y,z) значения, а для остальных переменных в этих случаях используются значения по умолчанию: z=0, w=1. Число в названии команды соответствует числу явно задаваемых значений, а последующий символ - их типу.

Координатные оси расположены так, что точка (0,0) находится в левом нижнем углу экрана, ось ***x*** направлена влево, ось ***y***- вверх, а ось ***z***- из экрана. Это расположение осей мировой системы координат, в которой задаются координаты вершин объекта.

Чтобы задать какую-нибудь фигуру в OpenGL используется понятие примитивов, к которым относятся точки, линии, связанные или замкнутые линии, треугольники и так далее. Задание примитива происходит внутри командных скобок:

glBegin(mode: GLenum)

…

glEnd

Параметр ***mode*** определяет тип примитива, который задается внутри и может принимать следующие значения:

- **GL_POINTS** каждая вершина задает координаты некоторой точки.

- **GL_LINES** каждая отдельная пара вершин определяет отрезок; если задано нечетное число вершин, то последняя вершина игнорируется.
- **GL_LINE_STRIP** каждая следующая вершина задает отрезок вместе с предыдущей.
- **GL_LINE_LOOP** отличие от предыдущего примитива только в том, что последний отрезок определяется последней и первой вершиной, образуя замкнутую ломаную.
- **GL_TRIANGLES** каждая отдельная тройка вершин определяет треугольник; если задано не кратное трем число вершин, то последние вершины игнорируются.
- **GL_TRIANGLE_STRIP** каждая следующая вершина задает треугольник вместе с двумя предыдущими.
- **GL_TRIANGLE_FAN** треугольники задаются первой и каждой следующей парой вершин (пары не пересекаются).
- **GL_QUADS** каждая отдельная четверка вершин определяет четырехугольник; если задано не кратное четырем число вершин, то последние вершины игнорируются.
- **GL_QUAD_STRIP** четырехугольник с номером n определяется вершинами с номерами 2n-1, 2n, 2n+2, 2n+1.
- **GL_POLYGON** последовательно задаются вершины выпуклого многоугольника.

Для задания текущего цвета вершины используются команды:

glColor[3 4][b s i f](components: GLtype)

glColor[3 4][b s i f]v(components: ^GLtype)

Первые три параметра задают R, G, B компоненты цвета, а последний параметр определяет alpha-компоненту, которая задает уровень прозрачности объекта. Разным вершинам можно назначать различные цвета и тогда будет проводиться линейная интерполяция цветов по поверхности примитива.

Задание на выполнение:

Заполнить структуру **PixelFormatDescriptor**. Согласно варианту задания построить несколько примитивов OpenGl.

Лабораторная работа №4 «Реализация аффинных преобразований средствами OpenGL»

Цель работы: Получить навыки масштабирования, поворота и переноса двумерных объектов OpenGL.

Для задания различных преобразований объектов сцены в OpenGL используются операции над матрицами. Видовая матрица определяет

преобразования объекта в мировых координатах, такие как параллельный перенос, изменение масштаба и поворот.

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. 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.

glTranslate[f d](x, y, z: GLtype) производит перенос объекта, прибавляя к координатам его вершин значения своих параметров.

glRotate[f d](angle, x, y, z: GLtype) производит поворот объекта против часовой стрелки на угол angle (измеряется в градусах) вокруг вектора (x,y,z).

glScale[f d](x, y, z: GLtype) производит масштабирование объекта (сжатие или растяжение), домножая соответствующие координаты его вершин на значения своих параметров.

Часто нужно сохранить содержимое текущей матрицы для дальнейшего использования, для чего используют команды **GlPushMatrix, glPopMatrix -** записывают и восстанавливают текущую матрицу из стека.

В случае если надо повернуть один объект сцены, а другой оставить неподвижным, удобно сначала сохранить текущую видовую матрицу в стеке командой glPushMatrix(), затем вызвать glRotate..() с нужными параметрами, описать примитивы, из которых состоит этот объект, а затем восстановить текущую матрицу glPopMatrix().

Задание на выполнение:

1. Применить к фигуре, построенной на лабораторной работе №3, преобразования переноса, поворота и масштабирования.
2. Организовать работу с преобразованиями изображения с использованием событий форм OnFormKeyDown или OnMouseMove.
3. Применить команды сохранения и восстановления текущего положения: glPushMatrix, glPopMatrix.
4. Организовать вращение изображение относительно оси Z по таймеру.
5. Организовать вывод одного из двумерных примитивов в цикле.

Лабораторная работа №5 «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.

Лабораторная работа №6 «Освещение и свойства материалов»

Цель работы: Получить навыки работы со свойствами материала и различными источниками света.

Для создания реалистических изображений необходимо определить свойства самого объекта и свойства среды, в которой он находится. Первая группа свойств включает в себя параметры материла, из которого сделан объект, способы нанесения текстуры на его поверхность, степень прозрачности объекта. Ко второй группе можно отнести количество и свойства источников света, уровень прозрачности среды. Все эти свойства можно задавать, используя соответствующие команды OpenGL.

Свойства материала

Для задания параметров текущего материала используются команды

glMaterial[i f](face, pname: GLenum, param: GLtype)

glMaterial[i f]v(face, pname: GLenum, params: ^GLtype)

С их помощью можно определить рассеянный, диффузный и зеркальный тип материала, а также степень зеркального отражения и интенсивность излучения света, если объект должен светиться. Какой именно параметр будет определяться значением param, зависит от значения pname:

- **GL_AMBIENT** параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют рассеянный цвет материала (цвет материала в тени). Значение по умолчанию: (0.2, 0.2, 0.2, 1.0).
- **GL_DIFFUSE** параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет диффузного отражения материала. Значение по умолчанию:(0.8, 0.8, 0.8, 1.0).
- **GL_SPECULAR** параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет зеркального отражения материала. Значение по умолчанию: (0.0, 0.0, 0.0, 1.0).
- **GL_SHININESS** параметр params должен содержать одно целое или вещественное значение в диапазоне от 0 до 128, которое определяет степень зеркального отражения материала. Значение по умолчанию: 0.
- **GL_EMISSION** параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют интенсивность излучаемого света материала. Значение по умолчанию: (0.0, 0.0, 0.0, 1.0).
- **GL_AMBIENT_AND_DIFFUSE** эквивалентно двум вызовам команды **glMaterial**..() со значением pname **GL_AMBIENT** и **GL_DIFFUSE** и одинаковыми значениями params.

В большинстве моделей учитывается диффузный и зеркальный отраженный свет; первый определяет естественный цвет объекта, а второй - размер и форму бликов на его поверхности.

Параметр face определяет тип граней, для которых задается этот материал и может принимать значения **GL_FRONT**, **GL_BACK** или **GL_FRONT_AND_BACK**.

Если в сцене материалы объектов различаются лишь одним параметром, рекомендуется сначала установить нужный режим, вызвав **glEnable**() с параметром **GL_COLOR_MATERIAL**, а затем использовать команду

glColorMaterial(face, pname: GLenum)

где параметр face имеет аналогичный смысл, а параметр pname может принимать все перечисленные значения. После этого, значения выбранного с помощью pname свойства материала для конкретного объекта (или вершины) устанавливается вызовом команды **glColor**..(), что позволяет избежать вызовов более ресурсоемкой команды **glMaterial**..() и повышает эффективность программы.

Источники света

Добавить в сцену источник света можно с помощью команд

glLight[i f](light, pname: GLenum, param: GLfloat)

glLight[i f]v(light, pname: GLenum, params: ^GLfloat)

Параметр light однозначно определяет источник, и выбирается из набора специальных символических имен вида **GL_LIGHT*i***, где i должно лежать в диапазоне от 0 до **GL_MAX_LIGHT**, которое не превосходит восьми.

Оставшиеся два параметра имеют аналогичный смысл, что и в команде **glMaterial**..(). Рассмотрим их назначение:

- **GL_SPOT_EXPONENT** параметр param должен содержать целое или вещественное число от 0 до 128, задающее распределение интенсивности света. Этот параметр описывает уровень сфокусированности источника света. Значение по умолчанию: 0 (рассеянный свет).
- **GL_SPOT_CUTOFF** параметр param должен содержать целое или вещественное число между 0 и 90 или равное 180, которое определяет максимальный угол разброса света. Значение этого параметра есть половина угла в вершине конусовидного светового потока, создаваемого источником. Значение по умолчанию: 180 (рассеянный свет).
- **GL_AMBIENT** параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет фонового освещения. Значение по умолчанию: (0.0, 0.0, 0.0, 1.0).
- **GL_DIFFUSE** параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет

диффузного освещения. Значение по умолчанию: (1.0, 1.0, 1.0, 1.0) для **LIGHT0** и (0.0, 0.0, 0.0, 1.0) для остальных.

- **GL_SPECULAR** параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет зеркального отражения. Значение по умолчанию: (1.0, 1.0, 1.0, 1.0) для **LIGHT0** и (0.0, 0.0, 0.0, 1.0) для остальных.
- **GL_POSITION** параметр params должен содержать четыре целых или вещественных, которые определяют положение источника света. Если значение компоненты w равно 0.0, то источник считается бесконечно удаленным и при расчете освещенности учитывается только направление на точку (x,y,z), в противном случае считается, что источник расположен в точке (x,y,z,w). Значение по умолчанию: (0.0, 0.0, 1.0, 0.0).
- **GL_SPOT_DIRECTION** параметр params должен содержать четыре целых или вещественных числа, которые определяют направление света. Значение по умолчанию: (0.0, 0.0, -1.0, 1.0).

При изменении положения источника света следует учитывать следующие факты: если положение задается командой **glLight**..() перед определением ориентации взгляда (командой **glLookAt**()), то будет считаться, что источник находится в точке наблюдения. Если положение устанавливается между заданием ориентации и преобразованиями видовой матрицы, то оно фиксируется и не зависит от видовых преобразований. В последнем случае, когда положение задано после ориентации и видовой матрицы, его положение можно менять, устанавливая как новую ориентацию наблюдателя, так и меняя видовую матрицу.

Для использования освещения сначала надо установить соответствующий режим вызовом команды **glEnable**(**GL_LIGHTING**), а затем включить нужный источник командой **glEnable**(**GL_LIGHT*n***).

Модель освещения

В OpenGL используется модель освещения Фонга, в соответствии с которой цвет точки определяется несколькими факторами: свойствами материала и текстуры, величиной нормали в этой точке, а также положением источника света и наблюдателя. Для корректного расчета освещенности в точке надо использовать единичные нормали, однако команды типа **glScale**..(), могут изменять длину нормалей. Чтобы это учитывать, используется уже упоминавшийся режим нормализации нормалей, который включается вызовом команды **glEnable**(**GL_NORMALIZE**).

Для задания глобальных параметров освещения используются команды
glLightModel[i f](pname, param: GLenum)
glLightModel[i f]v(pname: GLenum, const params: ^GLtype)

Аргумент pname определяет, какой параметр модели освещения будет настраиваться и может принимать следующие значения:

- **GL_LIGHT_MODEL_LOCAL_VIEWER** параметр param должен быть булевским и задает положение наблюдателя. Если он равен FALSE, то направление обзора считается параллельным оси -z, вне зависимости от положения в видовых координатах. Если же он равен TRUE, то наблюдатель находится в начале видовой системы координат. Это может улучшить качество освещения, но усложняет его расчет. Значение по умолчанию: FALSE.
- **GL_LIGHT_MODEL_TWO_SIDE** параметр param должен быть булевским и управляет режимом расчета освещенности как для лицевых, так и для обратных граней. Если он равен FALSE, то освещенность рассчитывается только для лицевых граней. Если же он равен TRUE, расчет проводится и для обратных граней. Значение по умолчанию: FALSE .
- **GL_LIGHT_MODEL_AMBIENT** параметр params должен содержать четыре целых или вещественных числа, которые определяют цвет фонового освещения даже в случае отсутствия определенных источников света. Значение по умолчанию:(0.2, 0.2, 0.2,1.0).

Задание на выполнение:

Согласно варианту задания построить трехмерную сцену с реализацией переопределения свойства материала объектов и изменением положения источника света.

Лабораторная работа №7 «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}
 quadGenus := gluNewQuadric;
 gluQuadricDrawStyle(quadGenus, GLU_FILL); //Стиль визуализации
 glNewList (1, GL_COMPILE);
        glMaterialfv(GL_FRONT,      GL_AMBIENT_AND_DIFFUSE,
@ColorGenus);
 glTranslatef(0.0 , -0.4, 0.0);
 gluCylinder (quadGenus, 0.25, 0.0, 0.8, 20, 20);
 glEndList;
end;

procedure FormDestroy(Sender: TObject);
begin
 gluDeleteQuadric(quadGenus);
 …
 …
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:

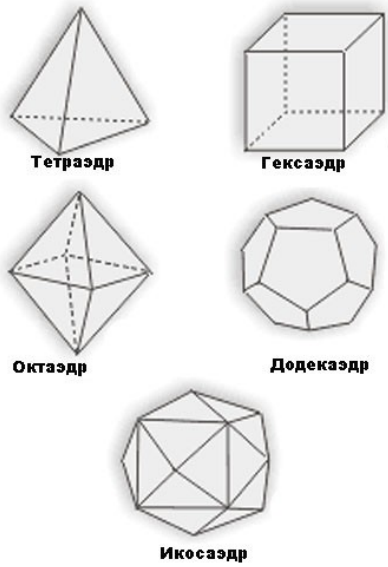
- **GLU_FILL** – отображение цельных, примитивов;
- **GLU_LINE** – примитивы отображаются набором линий;
- **GLU_SILHOUETTE** – примитивы отображаются как силуэт;
- **GLU_POINT** – примитивы состоят из точек.

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.

Лабораторная работа №8 «Наложение текстуры»

Цель работы: Получить навыки наложения текстур.

Наложение текстуры на поверхность объектов сцены повышает ее реалистичность, однако при этом надо учитывать, что этот процесс требует

значительных вычислительных затрат. Под текстурой будем понимать некоторое изображение, которое надо определенным образом нанести на объект. Для этого следует выполнить следующие этапы:

- выбрать изображение и преобразовать его к нужному формату
- загрузить изображение в память
- определить, как текстура будет наноситься на объект и как она будет с ним взаимодействовать.

Рассмотрим каждый из этих этапов.

Подготовка текстуры

Принятый в OpenGL формат хранения изображений отличается от стандартного формата Windows DIB только тем, что компоненты (R,G,B) для каждой точки хранятся в прямом порядке, а не в обратном и выравнивание задается программистом. Считывание графических данных из файла и их преобразование можно проводить и вручную, чуть ниже в примере мы продемонстрируем вам два способа загрузки текстуры из файлов средствами Delphi

При создании образа текстуры в памяти следует учитывать следующие требования.

Во-первых, размеры текстуры, как по горизонтали, так и по вертикали должны представлять собой степени двойки. Это требование накладывается для компактного размещения текстуры в памяти и способствует ее эффективному использованию. Использовать только текстуры с такими размерами конечно неудобно, поэтому перед загрузкой их надо преобразовать. Изменение размеров текстуры проводится с помощью команды

gluScaleImage(format: GLenum, widthin, heightin: GLint, typein: GLenum, const ^datain, widthout, heightout GLint, typeout: GLenum, ^dataout).

В качестве значения параметра format обычно используется значение **GL_RGB** или **GL_RGBA**, определяющее формат хранения информации. Параметры widthin, heightin, widhtout, heightout определяют размеры входного и выходного изображений, а с помощью typein и typeout задается тип элементов массивов, расположенных по адресам datain и dataout. Результат своей работы функция заносит в область памяти, на которую указывает параметр dataout.

Во-вторых, надо предусмотреть случай, когда объект по размерам значительно меньше наносимой на него текстуры. Чем меньше объект, тем меньше должна быть наносимая на него текстура и поэтому вводится понятие уровней детализации текстуры. Каждый уровень детализации задает некоторое изображение, которое является, как правило, уменьшенной в два раза копией оригинала. Такой подход позволяет улучшить качество нанесения текстуры на объект. Например, для изображения размером $2^m x 2^n$ можно построить max(m,n) +1 уменьшенных изображений, соответствующих различным уровням детализации.

Эти два этапа создания образа текстуры в памяти можно провести с помощью команды:

gluBuild2DMipmaps(target: GLenum, components, width, height: GLint, format, type: GLenum, const ^data),

где параметр target должен быть равен **GL_TEXTURE_2D**, components определяет количество цветовых компонент текстуры, которые будут использоваться при ее наложении и может принимать значения от 1 до 4 (1-только красный,2-красный и alpha, 3-красный, синий, зеленый, 4-все компоненты).

Параметры width, height, data определяют размеры и расположение текстуры соответственно, а format и type имеют аналогичный смысл, что и в команде **gluScaleImage**().

В OpenGL допускается использование одномерных текстур, то есть размера 1xN, однако это всегда надо указывать, используя в качестве значения target константу **GL_TEXTURE_1D**. Существует одномерный аналог рассматриваемой команды- **gluBuild1DMipmaps**(), который отличается от двумерного отсутствием параметра height.

При использовании в сцене нескольких текстур, в OpenGL применяется подход, напоминающий создание списков изображений. Вначале, с помощью команды

glGenTextures(n^ GLsizei, textures: ^GLuint)

надо создать n идентификаторов для используемых текстур, которые будут записаны в массив textures. Перед началом определения свойств очередной текстуры следует вызвать команду

glBindTexture(target: GLenum, texture: GLuint)

где target может принимать значения **GL_TEXTURE_1D** или **GL_TEXTURE_2D**, а параметр texture должен быть равен идентификатору той текстуры, к которой будут относиться последующие команды. Для того, чтобы в процессе рисования сделать текущей текстуру с некоторым идентификатором, достаточно опять вызвать команду **glBindTexture**() с соответствующим значением target и texture. Таким образом, команда **glBindTexture**() включает режим создания текстуры с идентификатором texture, если такая текстура еще не создана, либо режим ее использования, то есть делает эту текстуру текущей.

Методы наложения текстуры

При наложении текстуры, как уже упоминалось, надо учитывать случай, когда размеры текстуры отличаются от размеров объекта, на который она накладывается. При этом возможно как растяжение, так и сжатие изображения, и то, как будут проводиться эти преобразования, может серьезно повлиять на качество построенного изображения. Для определения положения точки на текстуре используется параметрическая система координат (s,t), причем значения s и t находятся в отрезке [0,1]. Для изменения различных параметров текстуры применяются команды:

glTexParameter[if](target, pname, param: GLEnum)

glTexParameter[if]v(target, pname:Glenum, params: ^GLEnum)

При этом target имеет аналогичный смысл, что и раньше, pname определяет, какое свойство будем менять, а с помощью param или params устанавливается новое значение. Возможные значения pname:

- **GL_TEXTURE_MIN_FILTER** параметр param определяет функцию, которая будет использоваться для сжатия текстуры. При значении **GL_NEAREST** будет использоваться один (ближайший), а при значении **GL_LINEAR** четыре ближайших элемента текстуры. Значение по умолчанию: **GL_LINEAR**.
- **GL_TEXTURE_MAG_FILTER** параметр param определяет функцию, которая будет использоваться для увеличения (растяжения) текстуры. При значении **GL_NEAREST** будет использоваться один (ближайший), а при значении **GL_LINEAR** четыре ближайших элемента текстуры. Значение по умолчанию: **GL_LINEAR**.
- **GL_TEXTURE_WRAP_S** параметр param устанавливает значение координаты s, если оно не входит в отрезок [0,1]. При значении **GL_REPEAT** целая часть s отбрасывается, и в результате изображение размножается по поверхности. При значении **GL_CLAMP** используются краевые значения: 0 или 1, что удобно использовать, если на объект накладывается один образ. Значение по умолчанию: **GL_REPEAT**.
- **GL_TEXTURE_WRAP_T** аналогично предыдущему значению, только для координаты t.

Использование режима **GL_NEAREST** значительно повышает скорость наложения текстуры, однако при этом снижается качество, так как в отличие от **GL_LINEAR** интерполяция не производится.

Для того, чтобы определить, как текстура будет взаимодействовать с материалом, из которого сделан объект, используются команды

glTexEnv[i f](target, pname, param: GLenum)

glTexEnv[i f]v(target, pname: Glenum, params: ^GLtype)

Параметр target должен быть равен **GL_TEXTURE_ENV**, а в качестве pname рассмотрим только одно значение **GL_TEXTURE_ENV_MODE**, которое применяется наиболее часто. Параметр param может быть равен:

- **GL_MODULATE** конечный цвет находится как произведение цвета точки на поверхности и цвета соответствующей ей точки на текстуре.
- **GL_REPLACE** в качестве конечного цвета используется цвет точки на текстуре.
- **GL_BLEND** конечный цвет находится как сумма цвета точки на поверхности и цвета соответствующей ей точки на текстуре с учетом их яркости.

Координаты текстуры

Перед нанесением текстуры на объект осталось установить соответствие между точками на поверхности объекта и на самой текстуре. Задавать это соответствие можно двумя методами: отдельно для каждой вершины или сразу для всех вершин, задав параметры специальной функции отображения.

Первый метод реализуется с помощью команд

glTexCoord[1 2 3 4][s i f d](coord: type)

glTexCoord[1 2 3 4][s i f d]v(coord: ^type)

Чаще всего используется команды вида **glTexCoord2**..(s, t: type), задающие текущие координаты текстуры. Вообще, понятие текущих координат текстуры аналогично понятиям текущего цвета и текущей нормали, и является атрибутом вершины. Однако даже для куба нахождение соответствующих координат текстуры является довольно трудоемким занятием, поэтому в библиотеке GLU помимо команд, проводящих построение таких примитивов, как сфера, цилиндр и диск, предусмотрено также наложение на них текстур. Для этого достаточно вызвать команду

gluQuadricTexture(quadObject: ^GLUquadricObj, textureCoords: GLboolean)

с параметром textureCoords равным **GL_TRUE**, и тогда текущая текстура будет автоматически накладываться на примитив.

Второй метод реализуется с помощью команд

glTexGen[i f d](coord, pname, param: GLenum)

glTexGen[i f d]v(coord, pname: GLenum, const params: ^GLtype)

Параметр coord определяет для какой координаты задается формула и может принимать значение **GL_S**, **GL_T**; pname определяет тип формулы и может быть равен **GL_TEXTURE_GEN_MODE**, **GL_OBJECT_PLANE**, **GL_EYE_PLANE**. С помощью params задаются необходимые параметры, а param может быть равен: **GL_OBJECT_LINEAR**, **GL_EYE_LINEAR**, **GL_SPHERE_MAP**.

Загрузка текстуры из BMP-файла

```
procedure TfmTex.BmpTexture;
var i, j: Integer;
begin
  Bitmap := TBitmap.Create;
  Bitmap.LoadFromFile('gold.bmp'); //загрузка текстуры из файла
  {--- заполнение битового массива ---}
  for i := 0 to 63 do
   for j := 0 to 63 do begin
    Bits [i, j, 0] := GetRValue(Bitmap.Canvas.Pixels[i,j]);
    Bits [i, j, 1] := GetGValue(Bitmap.Canvas.Pixels[i,j]);
    Bits [i, j, 2] := GetBValue(Bitmap.Canvas.Pixels[i,j]);
    Bits [i, j, 3] := 255;
```

```
    end;
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,
  64, 64,   //здесь задается размер текстуры
  0, GL_RGBA, GL_UNSIGNED_BYTE, @bits);
    glEnable(GL_TEXTURE_2D);
    glEnable(GL_TEXTURE_GEN_S);
    glEnable(GL_TEXTURE_GEN_T);
  end;
```

Задание на выполнение:

Согласно варианту задания наложить текстуру на двумерные и трехмерные объекты сцены.

Лабораторная работа №9 «Вывод текста»

Цель работы: Получить навыки работы с векторным и растровым типами текста OpenGL.

Реализация OpenGL под Windows содержит специфические команды, позволяющие обрабатывать TrueType шрифты и производить отображение текста на экране, представляя его в виде списков.

wglUseFontOutlines(dc: HDC, first, count, listBase: cardinal, deviation, extrusion: single, format: integer, lpgmf: ^TGLYPHMETRICSFLOAT)

Данная функция, принимая на вход контекст устройства Windows, строит списки изображений символов, используя текущий шрифт устройства. Параметрами этой функции являются: dc – контекст устройства; first – первый символ шрифта, заносимый в список; count – количество используемых символов шрифта; listBase – начальное смещение списков для всех обрабатываемых символов; deviation – качество аппроксимации шрифта (0 – отображать в оригинальном качестве); extrusion – ширина символа по оси Z; format – формат создания, может принимать следующие значения:

- **WGL_FONT_LINES** – шрифт строится контурными линиями
- **WGL_FONT_POLYGONS** – шрифт строится с помощью полигонов;

lpgmf – указатель на массив структур типа TGLYPHMETRICSFLOAT (GLYPHMETRICSFLOAT в Delphi 4), которые будут заполнены информацией о созданных списках.

После создания дисплейных списков символов, нам остается лишь обратится к функции **CallLists** для отображения текста на сцене OpenGL. Но для этого нам потребуется еще одна функция: **glListBase**(base: GLunit), которая устанавливает начальное смещение для всех вызываемых после нее списков.

Для создания растрового текст, к которым относятся битовые массивы, используется команда:

glBitmap(GLsizei width, GLsizei height, GLfloat xorigin, GLfloat yorigin, GLfloat xmove, GLfloatymove, const GLubyte* bitmap).

Параметры *width* и *height* задают, соответственно, ширину и высоту битового массива (в пикселях); *xorigin* и *yorigin* определяют точку окна, в которой будет расположен левый нижний угол битового массива; *xmove* и *ymove* показывают смещение, добавляемое к текущей позиции растра после вывода битового массива; *bitmap* указывает адрес, начиная с которого хранится собственно битовый образ.

При отображении битовый массив позиционируется относительно текущей позиции растра, и если она не определена, то битовый массив игнорируется. При определенной и разрешенной текущей позиции растра битовый массив начинается из точки с координатами

$$x_w = \lfloor x_r - xorigin \rfloor$$
$$y_w = \lfloor y_r - yorigin \rfloor$$

где x_r и y_r — координаты текущей позиции растра, а x_w и y_w — координаты образа в окне.

Команда формирует фрагменты для каждого пикселя, соответствующего 1 в битовом массиве. Помимо текущей позиции растра при формировании фрагмента используются текущие значения Z -координаты растра, цвет или индекс цвета, а также текущие координаты растра текстуры.

После того как битовый образ будет отображен в буфере кадра, к значениям координат текущей позиции растра добавляются соответствующие значения *xmove* и *ymove*. Остальные текущие параметры не изменяются.

Образ буквы F:

```
rasters : Array [0..23] of GLUByte =
($c0, $00, $c0, $00, $c0, $00, $c0, $00, $c0, $00,
  $ff, $00, $ff, $00, $c0, $00, $c0, $00, $c0, $00,
  $ff, $c0, $ff, $c0);
```

Задание на выполнение:

Согласно варианту задания вывести векторный и растровый типы текстов средствами библиотеки OpenGl.

Лабораторная работа №10 «Работа с туманом и прозрачностью»

Цель работы: Получить навыки применения спецэффектов: туман, прозрачность.

Для работы с туманом используется функция glEnable(GL_FOG).

Для контроля над туманом существуют функции

glFog[i f](pname: GLenum, param: Type)

glFog[i f]v(pname: GLenum, param: ^Type)

где свойство pname может принимать следующие значения:

- GL_FOG_MODE (значение param может принимать значение: GL_LINEAR, GL_EXP, GL_EXP2, определяя функцию вычисления цвета каждой точки изображения при наложении тумана)
- GL_FOG_DENSITY (значение param определяет плотность тумана. Значение по умолчанию 1.0)
- GL_FOG_START (ближайшая граница тумана. По умолчанию 0.0.)
- GL_FOG_END (задняя граница тумана. По умолчанию 1.0)
- GL_FOG_COLOR (параметр param является указателем на массив RGBA, определяющий цветовые составляющие тумана. Значение по умолчанию - (0,0,0,0))

При инициализации тумана задаем параметры:

1. включаем режим тумана;
2. задаем переменную, хранящую режим тумана;
3. задаем закон смешения тумана;
4. задаем цвет тумана.

Четвертая цветовая компонента – Alpha, отвечает за прозрачность отображаемой информации. Alpha-компонента может быть обработана двумя способами.

- вывод изображений с отсечением пикселей, не проходящих определенного порогового значения Alpha,
- наложение одного изображения на другое с использованием значения Alpha как уровня прозрачности выводимого изображения относительно уже находящегося в буфере либо наоборот.

За разрешения проверки порогового уровня Alpha отвечает команда glEnable(GL_ALPHA_TEST). После разрешения проверки, для каждого выводимого пикселя на экране будет выполняться проверка Alpha-компоненты по условию заданному с помощью

glAlphaFunc(func: GLenum, ref: GLclampf)

где ref – содержит некоторое пороговое значение, а func может иметь значение:

- GL_NEVER (не проходит)
- GL_LESS (проходит, если ref < alpha)
- GL_LEQUAL (проходит, если ref <= alpha)
- GL_GREATER (проходит, если ref > alpha)
- GL_GEQUAL (проходит, если ref >= alpha)
- GL_EQUAL (проходит, если ref = alpha)
- GL_NOTEQUAL (проходит, если ref <> alpha)

- GL_ALWAYS (всегда проходит)

В конечном результате, на экране будут отображены лишь пикселы, прошедшие тест.

Для включения режима отработки прозрачности, нам потребуется команда glEnable(GL_BLEND). Аналогично предыдущему случаю, при включении данного режима в действие вступает функция

glBlendFunc(sfactor, dfactor: GLenum)

где параметры sfactor и dfactor определяют соответственно способ формирования исходного (входного изображения) и конечного (отображаемой сцены) цветов. Всего существует 11 методов вычисления цветовых компонент, все они подробно рассмотрены в файле справочной системы, входящей в комплект Delphi. В данном случае нас интересует лишь два значения – GL_SRC_ALPHA для sfactor и GL_ONE_MINUS_SRC_ALPHA для dfactor. Этот способ работает при отображении сцены, объекты в которой расположены последовательно приближаясь к наблюдателю. В таком случае, при отображении очередного объекта мы, не изменив прозрачности уже созданной сцены, наложим на нее объект, учтя его Alpha-компоненту.

Задание на выполнение:

Согласно варианту задания создать трехмерную сцену с реализацией тумана и прозрачности.

Лабораторная работа №11 «Буфер трафарета»

Цель работы: Создать трехмерную сцену с использованием буфера трафарета. Например, три пересекающиеся фигуры или реализуем отверстия на поверхности с использованием буфера трафарета.

При выводе пикселей в буфер кадра иногда возникает необходимость выводить не все пиксели, а только некоторое подмножество, т.е. наложить трафарет (маску) на изображение. Для этого OpenGL предоставляет так называемый буфер трафарета (stencil buffer). Прежде чем поместить пиксель в буфер кадра, механизм визуализации OpenGL позволяет выполнить сравнение (тест) между заданным значением и значением в буфере трафарета. Если тест проходит, пиксель рисуется в буфере кадра.

Трафарет это двумерный массив целочисленных переменных. Каждому пикселю в окне соответствует один элемент массива. Использование буфера трафарета происходит в два этапа. Сначала вы его заполняете, потом, основываясь на его содержимом, рисуете ваши объекты. Буфер трафарета заполняется следующим образом. Вы делите окно вывода на зоны и каждой зоне присваиваете свое значение. Например, для нашей сцены, область, где нет ничего, будет заполнена нулями, область, где выведен куб, заполнена единицами и область, где видна сфера, двойками. Тест трафарета разрешается при помощи функции **glEnable** с параметром **GL_STENCIL_TEST**. Очищается буфер трафарета при помощи функции **glClear** с параметром **GL_STENCIL_BUFFER_BIT**. Заполнение буфера трафарета происходит при помощи следующих двух функций:

glStencilFunc(func: GLenum, ref: GLint, mask: GLuint)

glStencilOp(fail, zfail, zpass: GLenum)

Первая функция задает правило, по которому будет определяться, пройден тест трафарета или нет. Переменная func может принимать одно из следующих значений:

- **GL_NEVER** (не проходит)
- **GL_LESS** (проходит, если (ref **and** mask) < (stencil **and** mask)
- **GL_LEQUAL** (проходит, если (ref **and** mask) <= (stencil **and** mask)
- **GL_GREATER** (проходит, если (ref **and** mask) > (stencil **and** mask)
- **GL_GEQUAL** (проходит, если (ref **and** mask) >= (stencil **and** mask)
- **GL_EQUAL** (проходит, если (ref **and** mask) = (stencil **and** mask)
- **GL_NOTEQUAL** (проходит, если (ref **and** mask) <> (stencil **and** mask)
- **GL_ALWAYS** (всегда проходит)

Если тест трафарета не пройден, то фрагмент (пиксели) фигуры не прорисовываются в данном месте. Если тест пройден, то фигура рисуется. Вторая функция позволяет задать, как будет инициализироваться буфер трафарета. Параметры fail (тест трафарета не пройден), zfail (тест трафарета пройден, Z-буфера – нет) и zpass (пройдены оба теста, либо буфер глубины не используется) могут принимать одно из следующих значений:

- **GL_KEEP** (сохранить текущее значение в буфере трафарета)
- **GL_ZERO** (обнулить)
- **GL_REPLACE** (заменить на ref)
- **GL_INCR** (увеличить на единицу)
- **GL_DECR** (уменьшить на единицу)
- **GL_INVERT** (поразрядно инвертировать)

Буфер трафарета используется также при создании таких спецэффектов, как падающие тени, отражения, плавные переходы из одной картинки в другую.

Задание на выполнение:

Согласно варианту задания отобразить трехмерную сцену с использованием буфера трафарета.

Лабораторная работа №12 «Кривые Безье и NURBS-кривые»

Цель работы: Построить произвольную кривую Безье и NURBS-кривую.

Кубические кривые Безье и порции кубических поверхностей Безье довольно широко используются в задачах компьютерной графики и автоматизации проектирования.

Подход, предлагаемый OpenGL для изображения криволинейных поверхностей традиционен для компьютерной графики: задаются координаты небольшого числа опорных точек, определяющих вид искомой кривой или поверхности. В зависимости от способа расчета опорные точки могут лежать на получаемой кривой или поверхности, а могут и не располагаться на ней.

Рассмотрим наиболее распространенный вид кривых – кривые Безье (Bezier curves). Для построение кривой Безье средствами OpenGL в обработчике создания формы необходимо задать параметры *одномерного вычислителя*:

```
glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, @ctrlpoints);
glEnable(GL_MAP1_VERTEX_3);
```

Первый параметр – символическая константа, значение GL_MAP1_VERTEX_3 соответствует случаю, когда каждая контрольная точка представляет собой набор трех вещественных чисел. Значения второго и третьего аргументов команды определяют конечные точки интервала предварительного образа рассчитываемой кривой. Четвертый параметр «большой шаг», задает, сколько чисел содержится в считываемой порции данных. Последние два параметра команды – число опорных точек и указатель на массив опорных точек.

Для построения кривой можно использовать точки или отрезки: вместо команды, задающей вершину, вызывается команда *glEvalCoord*, возвращающая координаты рассчитанной кривой:

```
glBegin(GL_LINE_STRIP);
  For i := 0 to 30 do
   glEvalCoord1f(i/30.0);
glEnd;
```

Рассмотрим кубические B-сплайны, которые представляют собой усовершенствованную методику построения кубических кривых. Здесь снимается требование, чтобы формируемая кривая проходила через опорные точки, и накладывается новое – чтобы она проходила близко к ним. При этих условиях довольно просто обеспечить непрерывность не только самой составной кривой, но и ее первой и второй производных в точках сопряжения сегментов.

NURBS-кривые – один из классов B-сплайнов – рациональные B-сплайны, задаваемые на неравномерной сетке (Non-Uniform Rational B-Spline). Библиотека GLU предоставляет набор команд, позволяющих использовать данный класс поверхностей. Для работы с NURBS-кривыми в библиотеке GLU имеются переменные специального типа, используемые для идентификации объектов: *theNurb: GLUnurbsObj*. При создании окна объект, как обычно, создается: *theNurb := gluNewNurbsRenderer*. А в конце работы

приложения память, занимаемая объектом, высвобождается: *gluDeleteNurbsRenderer(theNurb)*.

Для манипулирования свойствами таких объектов предусмотрена специальная команда библиотеки: *gluNurbsProperty (theNurb, GLU_SAMPLING_TOLERANCE, 25.0)* – гладкость кривой задается допуском дискретизации (чем меньше это число, тем поверхность получается более гладкой).

В отличие от других объектов, NURBS-пкривые рассчитываются каждый раз заново. Построение кривой осуществляется командой:

gluNurbsCurve (theNurb, 8, @curveKnots, 3, @ctrlpoints, 4, GL_MAP1_VERTEX_3).

Первый аргумент – имя NURBS- объекта, вторым аргументом задается количество параметрических узлов кривой, третий аргумент – указатель на массив, хранящий значения этих узлов. Следующий параметр – смещение (сколько вещественных чисел содержится в порции данных), далее следует указатель на массив опорных точек. Последний аргумент равен порядку (степени) кривой плюс единица.

Задание на выполнение:

Согласно варианту задания реализовать сцену с использованием кривых Безье и NURBS- кривых.

Рекомендуемая литература

1. Перемитина Т.О. Компьютерная графика: Учебное пособие.– Томск: Томский межвузовский центр дистанционного образования, 2006. – 128 с.
2. Мураховский В.И. Компьютерная графика. Под ред. С.В. Симоновича. – М.: «АСТ-ПРЕСС СКД», 2002. –640 с.
3. Эйнджел Э. Интерактивная компьютерная графика. Вводный курс на базе OpenGl.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2001. – 592 с.
4. Порев В. Н. Компьютерная графика. – СПб.: БХВ-Петербург, 2004. – 432 с.
5. Люкшин Б.А. Инженерная и компьютерная графика: Учебное пособие. – Томск: Томский межвузовский центр дистанционного образования, 2004. – Ч.1: Вопросы теории компьютерной графики. – 141 с.
6. Ньюмен У., Спрулл Р. Основы интерактивной машинной графики. – М.: Мир, 1976.