Экзаменационная работа
(специальность “Технология программирования”)

Попов Владимир
группа П-92

Мультиплатформенная реализация трехмерной компьютерной графики

Руководитель работы: преп. Дединский И.Р.

Москва - 2002

ОГЛАВЛЕНИЕ

МУЛЬТИПЛАТФОРМЕННАЯ РЕАЛИЗАЦИЯ КОМПЬЮТЕРНОЙ ТРЕХМЕРНОЙ ГРАФИКИ

Попов Владимир, группа П-92

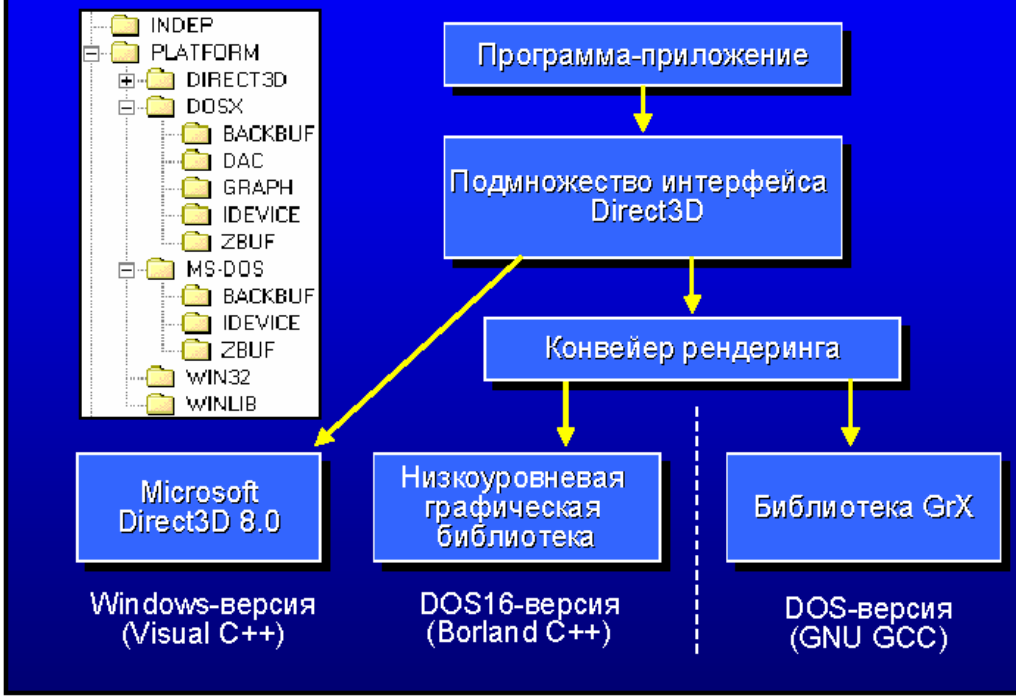
Цель проекта

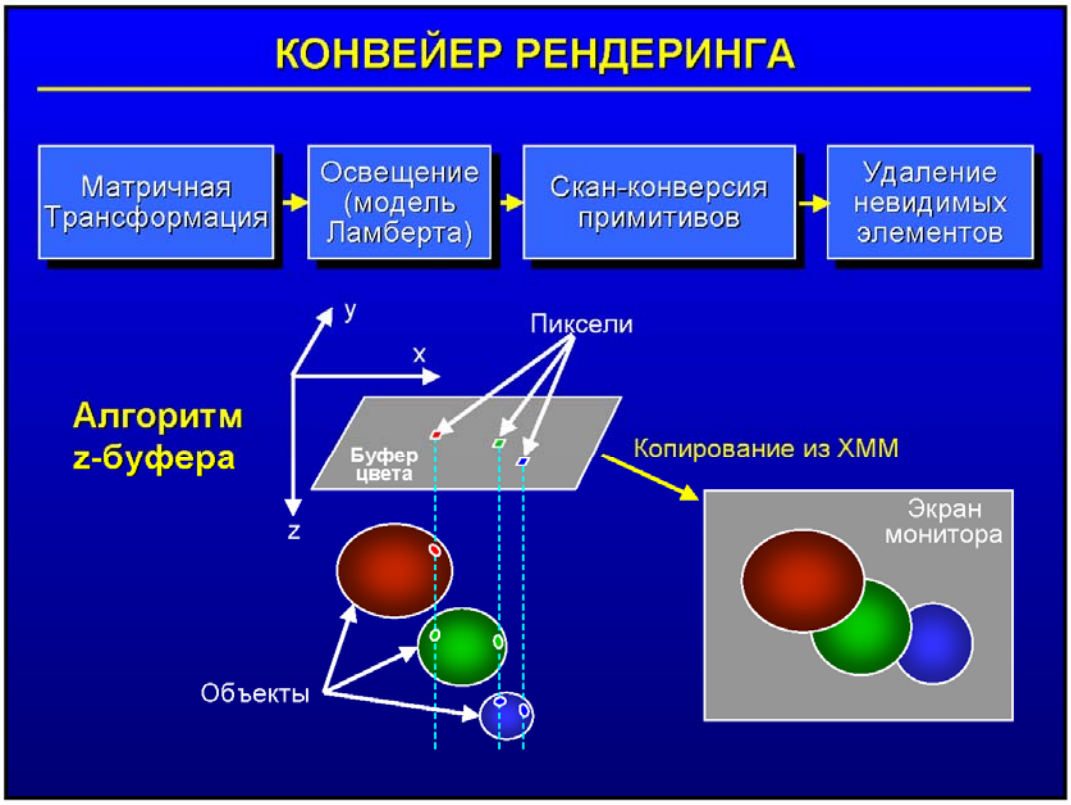
Создание набора библиотек 3D графики для работы под различные платформы (DOS16, DOS32 и Windows)

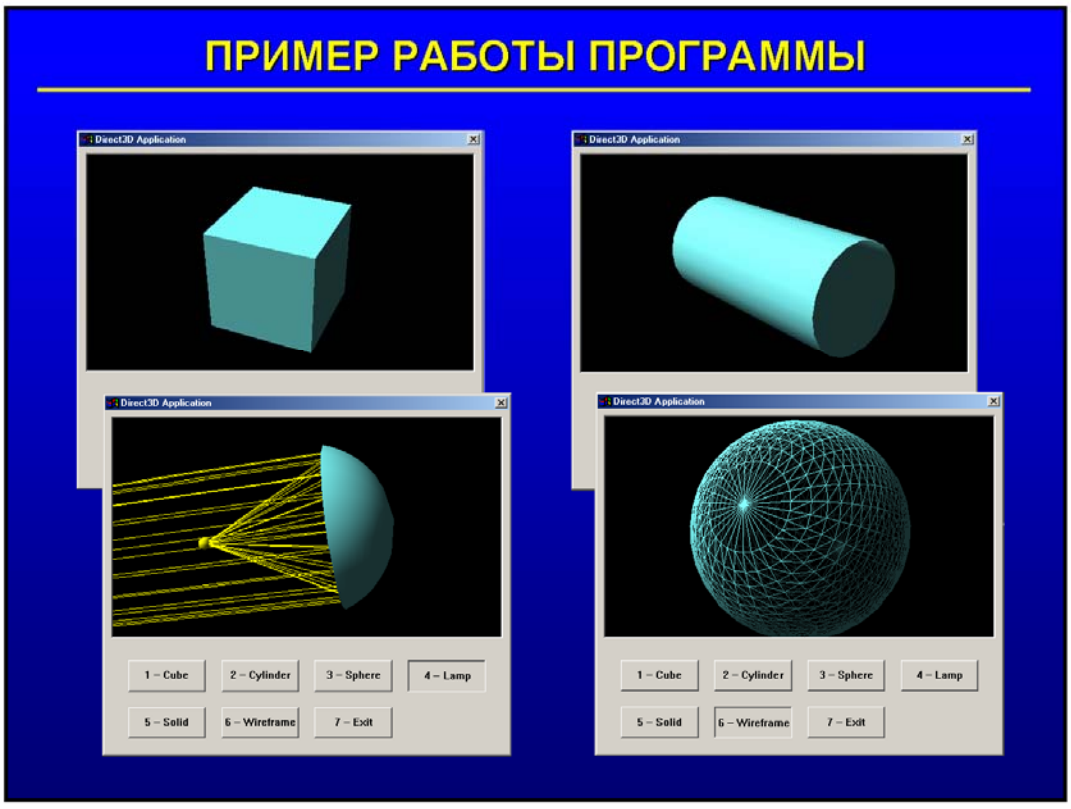
Задачи

- ✓ Планирование архитектуры для единой работы с несколькими компиляторами (Borland C++, GNU GCC, MS Visual C++)
- ✓ Создание конвейера рендеринга примитивов
- ✓ Эмуляция подмножества функций Windows API
- ✓ Эмуляция подмножества интерфейса Microsoft Direct3D 8.0
- ✓ Создание демонстрационной программы

АРХИТЕКТУРА БИБЛИОТЕКИ 3D ГРАФИКИ



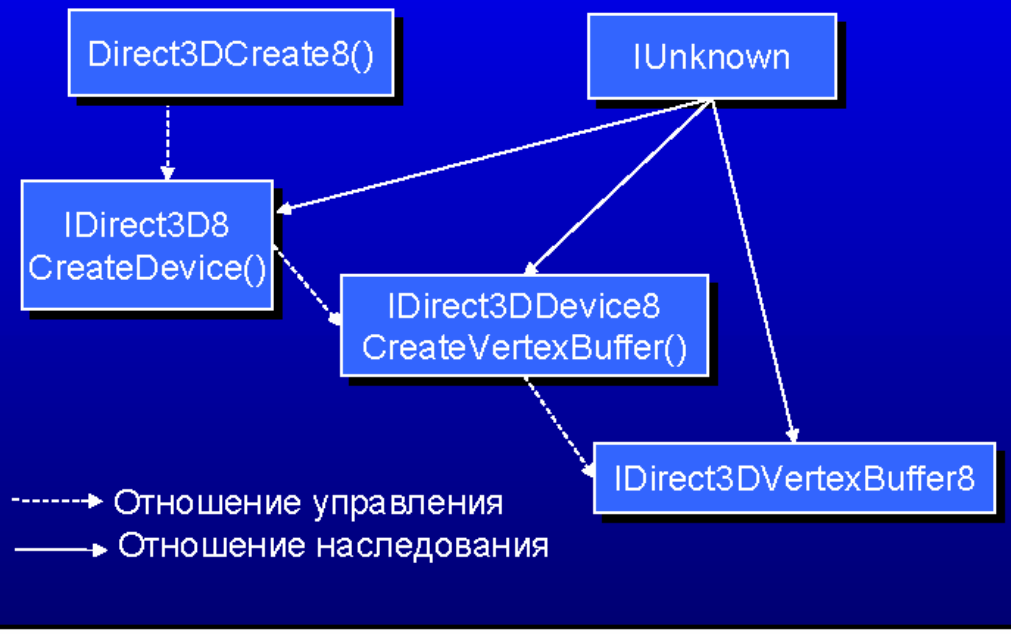


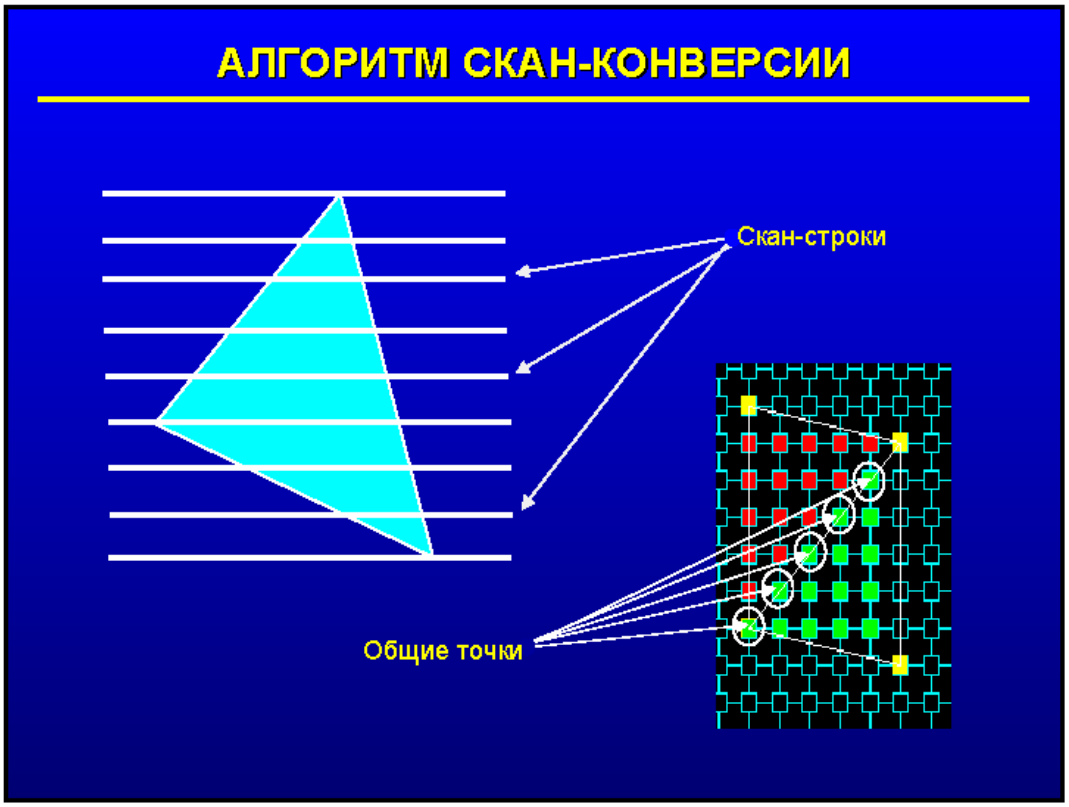


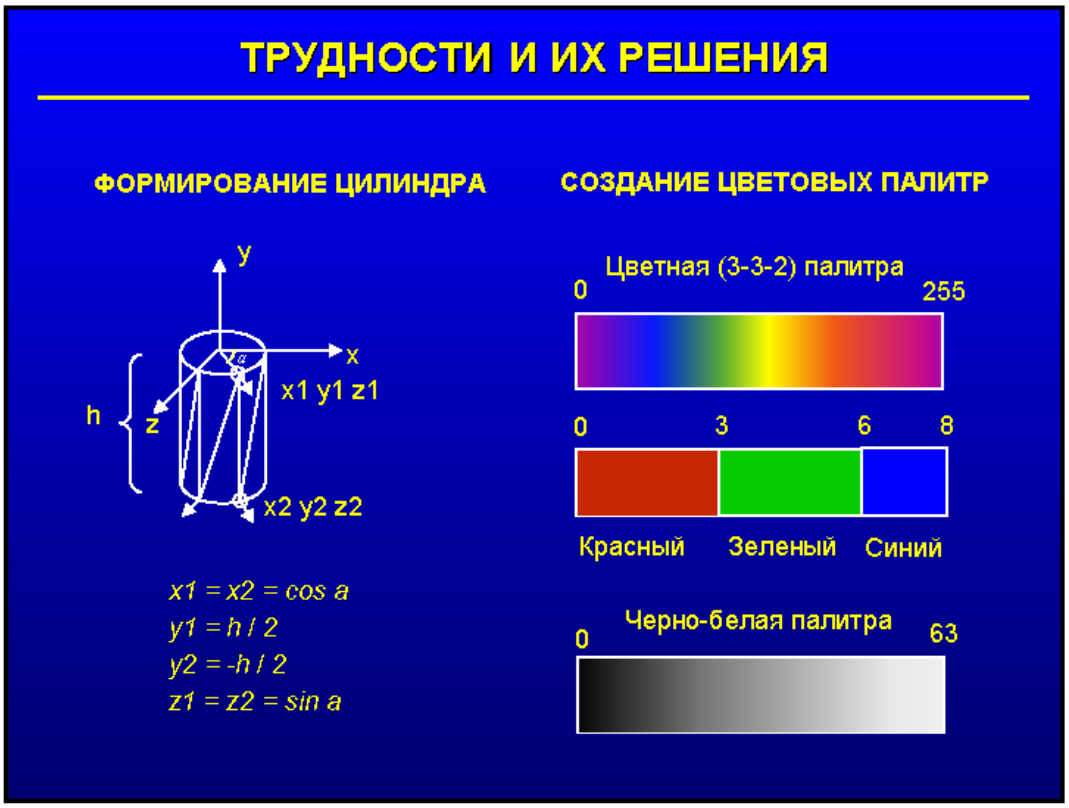
ВЫВОДЫ

- ✓ Были созданы версии под Win32, DOS16 и DOS32 для компиляторов Borland C++ 3.1, GNU GCC и MS Visual C++ 6.0
- ✓ Были эмулировано некоторое подмножество функций Windows API
- ✓ Было эмулировано некоторое подмножество интерфейса Microsoft Direct3D 8.0
- ✓ Был создан конвейер рендеринга
- ✓ Была создана платформно-независимая демонстрационная программа

ДИАГРАММА НАСЛЕДОВАНИЯ КЛАССОВ







Глава 1.
Вводная часть

1.1. Введение

1.1.1. Задача переносимости исходного кода

В настоящее время одной из наиболее актуальных задач программирования является переносимость исходного кода программы с одной платформы на другую. Это необходимо для сокращения переписываемого кода и более надежной разработки приложений, так как исходный код приложения пишется только единожды и затем его не нужно переписывать под другие платформы. Использование мультиплатформенного кода повышает надежность приложений, так как ошибку можно найти и исправить только в единой версии исходного кода, а затем перекомпилировать его под различные платформы.

1.1.2. Использование возможностей различных операционных систем

Каждая операционная система предоставляет пользователю и разработчику свои уникальные возможности, которых нет у других операционных систем. Операционная система Microsoft DOS предоставляет только основные возможности, такие как работа с командной строкой и работа с файлами. Но это система появилась достаточно давно и поэтому у нее низкие аппаратные требования, и она может присутствовать на любых машинах, от самых медленных, выпущенных в 80-ые годы, до современных высокопроизводительных рабочих станций. И MS-DOS не контролирует работу приложений, что позволяет программисту использовать все возможности машины. Также у MS-DOS отсутст-

вует многозадачность, поэтому приложения на основе данной операционной системы достигают максимальной производительности, так как не должны разделять процессорное время с другими приложениями. Операционная система Microsoft Windows на сегодняшний день является наиболее распространенной. В этой системе присутствует оконный интерфейс, что существенно упрощает работу с приложениями. Также в ней используется консолидирующая многозадачность, которая необходима для выполнения нескольких приложений одновременно.

1.1.3. Совместимость и поддержка возможностей современных операционных систем

Многие, если не все, производители аппаратного обеспечения выпускают драйвера для своих продуктов, ориентируясь на операционную систему Microsoft Windows. В операционной системе MS-DOS не удается воспользоваться функциями ускорения графики, которые присутствуют у всех современных видеокарт. В Windows это возможно, и для этого созданы библиотеки трехмерной графики, такие как OpenGL производства Silicon Graphics и Direct3D производства Microsoft. С увеличением возможностей видеокарт обновляются возможности и самих библиотек. В настоящее время, к выпуску готовится Direct3D версии 9.0 и OpenGL версии 2.0. Но бывают ситуации, когда необходима работа приложения на старых машинах под управлением MS-DOS. И тогда наиболее актуальной является задача всех функций API, необходимых для исполнения приложения.

1.2. Цель проекта

Цель проекта — создать набор библиотек, необходимых для возможности компиляции приложения под различными версиями операционных систем Mi-

crosoft Windows и Microsoft DOS. Необходимо было достигнуть совместимости только на уровне исходного кода.

1.3. Задачи проекта

1.3.1. Создание конвейера рендеринга

Под операционной системой MS-DOS необходимо выдавать на экран монитора то же изображение, что показывается под операционной системой Microsoft Windows с использованием Direct3D 8.0. Это требовало написания собственного конвейера рендеринга примитивов.

1.3.2. Эмуляция функций Windows API

Одной из задач была эмуляция функций Windows API. Поскольку используется полноэкранный режим, то большинство функций не выполняло никаких действий. Работа функций заключалась в проверке на нажатие клавиши и завершении программы при нажатии на произвольную клавишу.

1.3.3. Эмуляция интерфейса Microsoft Direct3D 8.0

Необходима эмуляция интерфейса Microsoft Direct3D 8.0. Для изображения трехмерных примитивов используется библиотека трехмерной графики Microsoft Direct3D 8.0. При помощи этого удается достичь аппаратной поддержки видео карт, что во много раз ускоряет процесс рендеринга. Для совместимости с MS-DOS необходимо создать те классы, которые используются Microsoft Direct3D 8.0 и наделить их соответствующей функциональностью.

1.3.4. Создание демонстрационной программы

В качестве мультиплатформенного кода должна быть создана программа, которая показывала на экране изображения различных объектов. Программа должна быть написана в стиле Windows API и API Direct3D версии 8.0.

1.3.5. Создание версий библиотеки, поддерживающих различные компиляторы

Часть библиотеки должна быть зависимой от компилятора и использовать его отличительные особенности. Планируется поддержка трех компиляторов: Borland C++ 3.1, GCC, Microsoft Visual C++ 6.0.

Глава 2.
Алгоритмы программы

1.1. Матричная алгебра

2.1.1. Терминология и обозначения

Матрицей А размера *m x n* называется набор *m x n* чисел — элементов матрицы

$$A=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \text{ или } A=\begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix},$$

Символ α_{ij} читается так: ”альфа-и-жи”.

Набор $\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{in}$ $(i = 1, \ldots m)$ называется *i*-ой строкой матрицы А = $(\alpha_{i1} \quad \alpha_{i2} \quad \alpha_{i3} \quad \alpha_{i4})$, а набор $\alpha_{1j,}\alpha_{2j},\ldots,\alpha_{mj}(j=1,\ldots n)$ называется *j*-м столбцом матрицы А:

$$\begin{pmatrix} \alpha_{1j} \\ \alpha_{2j} \\ \vdots \\ \alpha_{mj} \end{pmatrix}$$

Таким образом, данная матрица А имеет *m* строк и *n* столбцов, а элемент α_{ij} расположен в *i*-ой строке и в *j*-ом столбце матрицы А — в позиции (i, j). Числа *i* и *j* определяют расположение элемента α_{ij} в матрице А и являются как бы координатами этого элемента в прямоугольной таблице А.

Если размер матрицы известен, то часто пишут кратко

$$A = (\alpha_{ij})$$

Матрица размера $1 \times n$ называется просто строкой, а матрица размера $m \times 1$ — столбцом. В случае $m = n$ матрица

$$A = \begin{pmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \alpha_{n1} & \alpha_{n2} & \cdots & \alpha_{nn} \end{pmatrix}$$

называется квадратной матрицей порядка *n*. В частности, квадратной матрицей первого порядка является одноэлементная матрица $A = (\alpha_{11})$.

Набор элементов $\alpha_{11}, \alpha_{22}, \cdots, \alpha_{nn}$ образует главную диагональ матрицы A.

Матрица, все элементы которой равны нулю, называется нулевой, а квадратная матрица, на главной диагонали которой стоят единицы, а все остальные элементы равны нулю,

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 \end{pmatrix},$$

называется единичной. Для каждого размера $m \times n$ существует своя нулевая матрица, а для каждого числа *n* — своя единичная матрица порядка *n*.

Множество всех матриц размера $m \times n$ часто обозначают через $R_{m \times n}$. Введенное обозначение требует дополнительных пояснений (для определенности мы ограничиваемся здесь рассмотрением только матриц, элементами которых являются вещественные числа). Множество вещественных чисел принято обозначать через *R.* Отсюда и символ $R_{m \times n}$. С учетом этого обозначения матрицу можно записать так

$$A = (\alpha_{ij}) \in R_{m \times n}.$$

Матрицы $A = (\alpha_{ij})$ и $B = (\beta_{ij})$ называются равными, если они имеют одинаковый размер, и их элементы, находящиеся в одинаковых позициях, совпадают, т.е., $A \in R_{m \times n}, B \in R_{m \times n}$ и $\alpha_{ij} = \beta_{ij} (i = 1, \ldots, m; j = 1, \ldots, n)$. *Обозначение:* A = B.

2.1.2. Операции над матрицами

2.1.2.1. Сложение матриц

Пусть A и B — матрицы одного размера: $A=(\alpha_{ij})\in R_{m\times n}, B=(\beta_{ij})\in R_{m\times n}$.

Суммой матриц A и B называется матрица $C=(\gamma_{ij})\in R_{m\times n}$, элементы которой вычисляются по формуле $\gamma_{ij}=\alpha_{ij}+\beta_{ij}(i=1,\dots,m; j=1,\dots n)$.

Обозначение:.C = A + B.

2.1.2.2. Умножение матрицы на число

Произведением матрицы $A=(\alpha_{ij})\in R_{m\times n}$ на число λ называется матрица $B=(\beta_{ij})\in R_{m\times n}$, элементы которой вычисляются по формуле

$$\beta_{ij}=\lambda\alpha_{ij}(i=1,\dots,m; j=1,\dots n).$$

Обозначение: $B=\lambda A$.

Запишем эти операции подробнее:

$$\begin{pmatrix}\alpha_{11} & \cdots & \alpha_{1n}\\ \cdots & \cdots & \cdots\\ \alpha_{m1} & \cdots & \alpha_{mn}\end{pmatrix}+\begin{pmatrix}\beta_{11} & \cdots & \beta_{1n}\\ \cdots & \cdots & \cdots\\ \beta_{m1} & \cdots & \beta_{mn}\end{pmatrix}=\begin{pmatrix}\alpha_{11}+\beta_{11} & \cdots & \alpha_{1n}+\beta_{1n}\\ \cdots & \cdots & \cdots\\ \alpha_{m1}+\beta_{m11} & \cdots & \alpha_{mn}+\beta_{mn}\end{pmatrix},$$

$$\lambda\begin{pmatrix}\alpha_{11} & \cdots & \alpha_{1n}\\ \cdots & \cdots & \cdots\\ \alpha_{m1} & \cdots & \alpha_{mn}\end{pmatrix}=\begin{pmatrix}\lambda\alpha_{11} & \cdots & \lambda\alpha_{1n}\\ \cdots & \cdots & \cdots\\ \lambda\alpha_{m1} & \cdots & \lambda\lambda_{mn}\end{pmatrix}.$$

2.1.2.3. Умножение матриц

Пусть $A=(\alpha_{ik})$ и $B=(\beta_{ik})$ — квадратные матрицы порядка *n*. Произведением матрицы A на матрицу B называется матрица $C=(\gamma_{ij})\in R_{n\times n}$, элементы которой вычисляются по формуле

$$\gamma_{ij}=\alpha_{i1}\beta_{1j}+\dots+\alpha_{in}\beta_{nj}(i,j=1,\dots,n).$$

$$i\begin{pmatrix} \alpha_{11} & \cdots & \alpha_{1n} \\ \cdots & \cdots & \cdots \\ \boxed{\alpha_{i1}} & \cdots & \boxed{\alpha_{in}} \\ \cdots & \cdots & \cdots \\ \alpha_{n1} & \cdots & \alpha_{nn} \end{pmatrix} \bullet \begin{pmatrix} \beta_{11} & \vdots & \overset{j}{\boxed{\beta_{1j}}} & \vdots & \beta_{1n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \beta_{n1} & \vdots & \boxed{\beta_{nj}} & \vdots & \beta_{nn} \end{pmatrix} = \begin{pmatrix} \gamma_{11} & \overset{j}{\vdots} & \gamma_{1n} \\ \cdots & \boxed{\gamma_{ij}} & \cdots \\ \gamma_{n1} & \vdots & \gamma_{mn} \end{pmatrix}$$

2.1.2.4. Транспонирование матрицы

Матрица A называется транспонированной по отношению к матрице A^T:

$$A = \begin{pmatrix} \alpha_{11} & \alpha_{21} & \cdots & \alpha_{m1} \\ \alpha_{12} & \alpha_{22} & \cdots & \alpha_{m2} \\ \cdots & \cdots & \cdots & \cdots \\ \alpha_{1n} & \alpha_{2n} & \cdots & \alpha_{mn} \end{pmatrix}, \; A^T = \begin{pmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \alpha_{m1} & \alpha_{m2} & \cdots & \alpha_{mn} \end{pmatrix}.$$

2.1.3. Определитель матрицы

Свяжем с каждой квадратной матрицей число — определитель матрицы — по следующему правилу.

Будем считать, что определитель матрицы (α_{11}) первого порядка равен числу α_{11}.

Определителем матрицы второго порядка $\begin{pmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{pmatrix}$ называется число, равное $\alpha_{11}\alpha_{22} - \alpha_{12}\alpha_{21}$.

Обозначение:

$$\det\begin{pmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{pmatrix} = \begin{vmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{vmatrix} = \alpha_{11}\alpha_{22} - \alpha_{12}\alpha_{21}.$$

Определителем матрицы третьего порядка

$$\begin{pmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{pmatrix}$$

называется число, равное

$$\alpha_{11}\begin{vmatrix} \alpha_{22} & \alpha_{23} \\ \alpha_{32} & \alpha_{33} \end{vmatrix} - \alpha_{21}\begin{vmatrix} \alpha_{12} & \alpha_{13} \\ \alpha_{32} & \alpha_{33} \end{vmatrix} + \alpha_{31}\begin{vmatrix} \alpha_{12} & \alpha_{13} \\ \alpha_{22} & \alpha_{23} \end{vmatrix}.$$

Предположим теперь, что определители матриц, порядок которых меньше n, уже введены. Определителем матрицы n-го порядка

$$A = \begin{pmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \alpha_{n1} & \alpha_{n2} & \cdots & \alpha_{nn} \end{pmatrix}$$

называется число, равное

$$D = \alpha_{11}M_{11} - \alpha_{12}M_{12} + \ldots + (-1)^{n+1}\alpha_{n1}M_{n1}.$$

Здесь $M_{i1}(i = 1,\ldots,n)$ — определитель матрицы порядка $n - 1$:

$$\begin{pmatrix} \alpha_{12} & \cdots\cdots & \alpha_{1n} \\ \alpha_{22} & \cdots\cdots & \alpha_{2n} \\ \cdots\cdots & \cdots\cdots & \cdots\cdots \\ \alpha_{i-1,2} & \cdots\cdots & \alpha_{i-1,n} \\ \alpha_{i+1,2} & \cdots\cdots & \alpha_{i+1,n} \\ \cdots\cdots & \cdots\cdots & \cdots\cdots \\ \alpha_{n2} & \cdots\cdots & \alpha_{nn} \end{pmatrix}.$$

2.1.4. Алгебраическое дополнение матрицы

Минором M_{ij} элемента α_{ij} определителя Δ называется определитель, получаемый из данного путем вычеркивания элементов i-ой строки и j-го столбца, на пересечении которых находится этот элемент. Например, минором элемента α_{23} будет определитель

$$M_{23} = \begin{vmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{31} & \alpha_{32} \end{vmatrix}.$$

Алгебраическим дополнением A_{ij} элемента α_{ij} называется минор M_{ij} этого элемента, взятый со своим знаком, если сумма $i + j$ номеров строки и столбца, на пересечении которых расположен элемент α_{ij}, есть число четное, и с противоположным знаком, если это число нечетное:

$$A_{ij} = (-1)^{i+j} M_{ij}.$$

2.1.5. Обратная матрица

Квадратная матрица называется невырожденной, если определитель этой матрицы не равен нулю.

Пусть $A=(\alpha_{ij})$ — невырожденная матрица порядка *n*. Построим новую квадратную матрицу B порядка *n* по следующему правилу: в *i*-ю строку и *j*-й столбец матрицы B — в позицию (i,j) — помещается число, равное алгебраическому дополнению A_{ji} элемента α_{ji} матрицы A:

$$A=\begin{pmatrix} \alpha_{11} & \vdots & \alpha_{1n} \\ \cdots & \alpha_{ji} & \cdots \\ \alpha_{n1} & \vdots & \alpha_{nn} \end{pmatrix} j \Rightarrow B=\begin{pmatrix} A_{11} & \vdots & A_{n1} \\ \cdots & A_{ji} & \cdots \\ A_{1n} & \vdots & A_{nn} \end{pmatrix} i.$$
$$\qquad\qquad\quad i \qquad\qquad\quad j$$

Матрица B обладает следующим свойством:

$$AB=BA=\begin{pmatrix} |A| & & 0 \\ & \ddots & \\ 0 & & |A| \end{pmatrix}=|A|\bullet I$$

Докажем равенство $AB=|A|\bullet I$.

Элемент произведения AB, находящийся в позиции (i,j), вычисляется по формуле $\gamma_{ij}=\sum_{k=1}^{n}\alpha_{ik}A_{jk}$. При $i=j$ получаем разложение определителя матрицы A по *i*-ой строке: $\gamma_{ij}=\sum_{k=1}^{n}\alpha_{ik}A_{ik}=|A|.$

Матрица A^{-1} называется обратной к матрице A:

$$A^{-1}=\frac{1}{|A|}B=\begin{pmatrix} \frac{A_{11}}{|A|} & \cdots & \frac{A_{n1}}{|A|} \\ \cdots & \cdots & \cdots \\ \frac{A_{1n}}{|A|} & \cdots & \frac{A_{nn}}{|A|} \end{pmatrix}$$

2.2. Векторная алгебра

2.2.1. Определение вектора

Рассмотрим две точки A и B. По соединяющему их отрезку можно перемещаться в любом из двух противоположных направлений. Если считать, например, точку A начальной, а точку B конечной, то тогда получаем направленный отрезок AB, в другом случае — направленный отрезок BA. Направленные отрезки часто называют связанными или закрепленными векторами. На чертеже заданное направление указывается стрелкой.

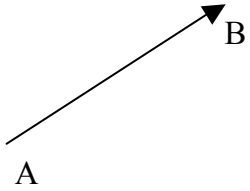


Рис. 1 Вектор

В случае, когда начальная и конечная точки совпадают, A = B, связанный вектор называется нулевым.

Свободные векторы — векторы, начальную точку которых можно выбирать произвольно, или, что то же самое, которые можно произвольно переносить параллельно самим себе. Свободный вектор $\vec{AB}$ однозначно определяется заданием связанного вектора AB. Если в качестве начальных выбирать лишь те точки, которые лежат на прямой, определяемой заданным (ненулевым) связанным вектором, то мы переходим к понятию скользящего вектора.

2.2.2. Линейные операции над векторами

2.2.2.1. Сложение векторов

Пусть заданы два вектора a и b. Возьмем какую-нибудь точку O и отложим от нее вектор a: $\vec{OA} = a$. От полученной точки A отложим вектор b: $\vec{AB} = b$. По-

лученный в результате вектор $\vec{OB}$ называется суммой векторов a и b и обозначается через a + b. Этот способ построения суммы векторов называется правилом треугольника.

Сложение векторов коммутативно, т.е. для любых векторов справедливо равенство $a+b=b+a$.

Если отложить векторы a и b от общей точки O и построить на них как на сторонах параллелограмм, то вектор $\vec{OB}$, идущий из общего начала O в противоположную вершину параллелограмма, будет их суммой a + b (или b + a). Этот способ построения суммы векторов называется правилом параллелограмма.

Пусть заданы три вектора, например, a, b и c. Отложим от произвольной точки O вектор a: $\vec{OA}=a$; от полученной точки A отложим вектор b: $\vec{AB}=b$; от точки B — вектор c: $\vec{BC}=c$. По определению суммы $\vec{OB}=a+b$ и $\vec{OC}=(a+b)+c$. С другой стороны, $\vec{AC}=b+c$ и, значит, $\vec{OC}=a+(b+c)$. Тем самым, для любых векторов a, b и c выполняется равенство $(a+b)+c=a+(b+c),$ т.е. сложение векторов ассоциативно. Опуская скобки, можно говорить о сумме трех векторов и записывать ее так: $a+b+c$.

Аналогично определяется сумма любого числа векторов: это есть вектор, который замыкает ломаную, построенную из заданных векторов. Приведенный способ сложения произвольного числа векторов называется правилом замыкающего ломаную.

2.2.2.2. Умножение вектора на число

Определение: Свободные векторы a и b называются коллинеарными, если определяющие их связанные векторы лежат на параллельных или совпадающих прямых.

Обозначение: a || b.

Если отложить коллинеарные векторы a и b от общей точки O, $\vec{OA} = a, \vec{OB} = b$, то точки O, A и B будут лежать на одной прямой. При этом возможны два случая: точки A и B располагаются на этой прямой: 1) по одну сторону от точки O, 2) по разные стороны. В первом случае векторы a и b называются одинаково направленными, а во втором — противоположно направленными.

Если векторы имеют равные длины и одинаково направлены, то они равны.

Пусть a — вектор, λ — вещественное число.

Определение: Произведением вектора a на число λ называется вектор b такой, что (1) $|b| = |\lambda| \cdot |a|$, и (2) векторы a и b одинаково (соответственно, противоположно) направлены, если $\lambda > 0$ (соответственно, $\lambda < 0$).

Обозначение: b = λ a.

Определение: Вектор, длина которого равна единице, называется единичным вектором, или ортом, и обозначается a^0, $|a^0| = 1$

Если $a \neq 0$, то вектор $a^0 = \frac{1}{|a|} a = \frac{a}{|a|}$ есть единичный вектор (орт) направления вектора a.

2.2.3. Координаты и компоненты вектора

Выберем в пространстве прямоугольную декартову систему координат. Обозначим через i, j, k единичные векторы (орты) положительных направлений осей Ox, Oy, Oz. Рассмотрим произвольный вектор a, начало которого лежит в начале координат O, а конец — в точке A. Проведем через точку A плоскости, перпендикулярные осям Ox, Oy, Oz. Это плоскости пересекут координатные оси в точках P, Q и R соответственно: $a = \vec{OP} + \vec{OQ} + \vec{OR}$.

Векторы $\vec{OP}, \vec{OQ} \, и \, \vec{OR}$ коллинеарны соответственно единичным векторам i, j, k, поэтому найдутся числа x, y, z такие, что $\vec{OP} = xi, \vec{OQ} = yj, \vec{OR} = zk.$

Формула называется разложением вектора a по векторам i, j, k. Указанным способом всякий вектор может быть разложен по векторам i, j, k.

Векторы i, j, k попарно ортогональны, и их длины равны единице. Тройку i, j, k называют ортонормированным (координатным) базисом (ортобазисом).

Можно показать, что для каждого вектора a разложение по базису i, j, k единственно, т.е. коэффициенты x, y, z в разложении вектора a по векторам i, j, k определены однозначно. Эти коэффициенты называются координатами вектора a. Они совпадают с координатами x, y, z точки A — конца вектора a. Мы пишем в этом случае $a = \{x, y, z\}$.

Эта запись означает, что свободный вектор a однозначно задается упорядоченной тройкой своих координат. Векторы xi, yj, zk, сумма которых равна вектору a, называются компонентами вектора a.

Из вышеизложенного следует, что два вектора $a = \{x_1, y_1, z_1\}$ и $b = \{x_2, y_2, z_2\}$ равны тогда и только тогда, когда соответственно равны их координаты, т.е.

$$a = b \Leftrightarrow \begin{cases} x_1 = x_2 \\ y_1 = y_2 \\ z_1 = z_2 \end{cases}$$

Радиусом-вектором точки M (x, y, z) называется вектор r = xi + yj + zk, идущий из начала координат O в точку M

2.2.3.1. Линейные операции над векторами в координатах

Пусть имеем два вектора $a = \{x_1, y_1, z_1\}$ и $b = \{x_2, y_2, z_2\}$, так что $a = x_1 i + y_1 j + z_1 k$, $b = x_2 i + y_2 j + z_2 k$. На основании правила сложения векторов имеем $a + b = (x_1 i + y_1 j + z_1 k) + (x_2 i + y_2 j + z_2 k) = (x_1 + x_2) i + (y_1 + y_2) j + (z_1 + z_2) k$, или $a + b = \{x_1 + x_2, y_1 + y_2, z_1 + z_2\}$ — при сложении векторов их координаты попарно складываются.

Аналогично получаем $a - b = \{x_1 - x_2, y_1 - y_2, z_1 - z_2\}$.

Далее, $\lambda a = \lambda x_1 i + \lambda y_1 j + \lambda z_1 k$ или $\lambda a = \{\lambda x_1, \lambda y_1, \lambda z_1\}$ — при умножении вектора на число все его координаты умножаются на это число.

2.2.4. Скалярное произведение векторов

Пусть имеем два вектора a и b.

Определение: Скалярным произведением вектора a на вектор b называется число, обозначаемое равенством $(a,b) = |a| \cdot |b| \cdot \cos\varphi = |a| \cdot |b| \cdot \cos(\hat{a,b})$, где φ, или в иной записи $(\hat{a,b})$, есть угол между векторами a и b.

Пусть векторы a и b заданы своими координатами в ортонормированном базисе i, j, k: $a = \{x_1, y_1, z_1\}$, $b = \{x_2, y_2, z_2\}$.

Рассмотрим скалярное произведение векторов a и b:

$$(a,b) = (x_1 i + y_1 j + z_1 k, x_2 i + y_2 j + z_2 k).$$

Учитывая, что $(i,j) = (i,k) = (j,k) = 0, \quad (i,i) = (j,j) = (k,k) = 1,$

получаем $(a,b) = x_1 x_2 + y_1 y_2 + z_1 z_2$.

То есть, если векторы a и b заданы своими координатами в ортонормированном базисе, то их скалярное произведение равно сумме произведений одноименных координат.

2.2.5. Векторное произведение векторов

Определение. Векторным произведением вектора a на вектор b называется вектор, обозначаемый символом [a, b] (или $a \times b$), такой, что

1) длина вектора [a, b] равна $|a| \cdot |b| \cdot \sin\varphi$, где φ — угол между векторами a и b;
2) вектор [a, b] перпендикулярен векторам a и b, т.е. перпендикулярен плоскости этих векторов;
3) вектор [a, b] направлен так, что из конца этого вектора кратчайший поворот от a к b виден происходящим против часовой стрелки.

Иными словами, векторы a, b и [a, b] образуют правую тройку векторов, т.е. расположены так, как большой, указательный и средний пальцы правой руки.

В случае, если векторы коллинеарны, будем считать, что [a, b] = 0.

2.3. Алгоритм скан-конверсии треугольника

Для рендеринга треугольника использовался алгоритм скан конверсии. Этот алгоритм позволяет исключить постановку точек смежных ребер двух соприкасающихся треугольников (рис 2). На данном рисунке видно, как решается проблема разделения общих точек. Алгоритм обрабатывает треугольник по строкам. Для вычисления координаты *x* используется уравнение линии. ($y = kx + b$). Если *x* координата точки содержит дробную часть, то координата округляется в большую сторону, если координата принадлежит левой грани треугольника, или в меньшую сторону, если координата принадлежит правой грани треугольника. Если точка принадлежит левой грани треугольника и при этом не имеет дробной части, то точка ставится. Если такая точка принадлежит правой грани треугольника, то точка не ставится. Если у треугольника есть нижняя грань, то точки, принадлежащие нижней грани, не ставятся. В процессе выполнения скан конверсии треугольника необходимо выяснять, какая из трех точек треугольника является верхней (имеет самую малую *y* координату), а какая нижней (имеет самую большую *y* координату). Может возникнуть такая ситуация, когда у треугольника есть верхняя грань. Тогда невозможно определить самую верхнюю точку. Тогда алгоритм рисует треугольник не с верхней точки до нижней, а с нижней точки до верхней. Также во время исполнения алгоритма происходит билинейная интерполяция цвета пикселей и *z*-координат вершин. Эти действия необходимы для модели закраски треугольника по Гуро и для реализации алгоритма *Z*-буфера.

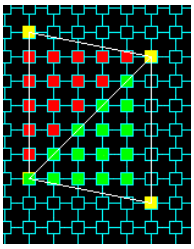


Рис. 2 Скан-конверсия

Ниже приведен алгоритм скан конверсии треугольника:

```
Для каждой строки пикселов:
  {
  Округлить_в_большую_сторону (Координата_начала_строки);
  Округлить_в_меньшую_сторону (Координата_конца_строки);

  Если Координата_конца_строки – целая, то
     Координата_конца_строки--;

  Рисовать_строку (Координата_начала_строки,
            Координата_конца_строки, Текущая трока);
  }
```

2.4. Алгоритм *Z*-буфера

В процессе рендеринга на экран необходимо ставить точки. Это нужно для появления на экране очередного треугольника. Но треугольник может быть частично или полностью скрыт. Необходимо решить проблему, ставить ли точки или нет. Эту проблему решает алгоритм *Z*-буфера. Идея этого алгоритма состоит в том, что у каждого пикселя есть *z*-координата. При постановке пикселя алгоритм проверяет, *z*-координата какой точки больше: той, которую уже поставили, или которую нужно ставить в текущий момент. Поскольку для реализации этого алгоритма требуется много памяти (например, для разрешения 1024x768x256 необходимо 1.5 мб RAM). Для работы с таким объемом памяти используется расширенная память — XMS (eXtended Memory Specification).

Ниже приведен алгоритм *z*-буфера:

```
Процедура Ставить_точку (x, y, z, цвет)
  {
  если старая z-координата > новая z-координата, то
     {
     ставь_точку (x, y, цвет);
     старая z-координата = новая z-координата
     }
  иначе
     /* точка лежит ниже какой-то поверхности,
        ничего не делаем */
  }
```

2.5. Использование двойной буферизации с использованием расширенной памяти

Приложение создает на экране анимированное изображение. Для этого требуется его постоянная перерисовка. При этом появляется проблема постоянного мерцания изображения на экране монитора. Одной из технологий, позволяющих решить данную проблему является технология двойной буферизации. Идея данной технологии состоит в том, что приложение выполняет все промежуточные действия (стирание изображения, его повторную перерисовку) не в видеопамяти, а в какой-либо области памяти, которая затем копируется на экран. Производители графических карт добавили такие возможности в свои продукты, но под управлением MS-DOS такая технология, которая использует несколько видеостраниц и называется page flipping, доступна на очень низких разрешениях (например, 320x240x256). А на более высоких приходится использовать собственные буфера, которые не работают достаточно быстро для качественной анимации. Но эта технология позволяет полностью решить проблему мерцания изображения и поэтому используется алгоритмами рендеринга изображения. Поскольку, как и алгоритм *z*-буфера, алгоритм двойной буферизации требует очень много памяти (например, для разрешения 1024x768x256 необходимо 768 кб RAM), для его реализации используется XMS.

2.6. Модель закраски и освещения

У каждой вершины есть позиция и нормаль. Нормаль — это перпендикуляр к поверхности. В сцене присутствуют источники света.

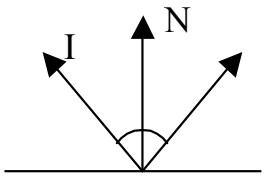


Рис. 3. К расчету освещения

Алгоритм рендеринга считает только диффузное или Ламбертово освещение. Пусть у нас есть два вектора: N – нормаль и I –направление на источник света. И нормаль, и направление на свет должны быть ортонормированными, т.е. их длина должна равняться единице. Интенсивность света равна скалярному произведению направления на источник света на нормаль.

$$P = I \cdot N$$

Такую модель предложил Ламберт, поэтому и модель назвали ламбертовой.

Существует три модели закраски треугольника: плоская, по Гуро и по Фонгу. Плоская модель закраски не может адекватно передать освещение объектов, так как треугольнику назначается один цвет, и не происходит плавных цветовых переходов по поверхности треугольника. Модель закраски по Фонгу наиболее реалистично передает эффекты освещения, так как при скан конверсии треугольника происходит интерполяция нормалей и освещение вычисляется в каждой точке. Это вычислительно дорого, и такая модель используется только в аппаратной реализации, которая присутствует лишь у наиболее дорогих видеокарт.

Если треугольник закрашивается по Гуро, то вычисляется цвет вершин треугольника, а затем происходит интерполяция цвета по площади треугольника:

$$Step = (color2-color1) / dy,$$

где *color2* и *color1* – цвета вершин треугольника, *dy* – расстояние по оси *y* от одной вершины треугольника до другой.

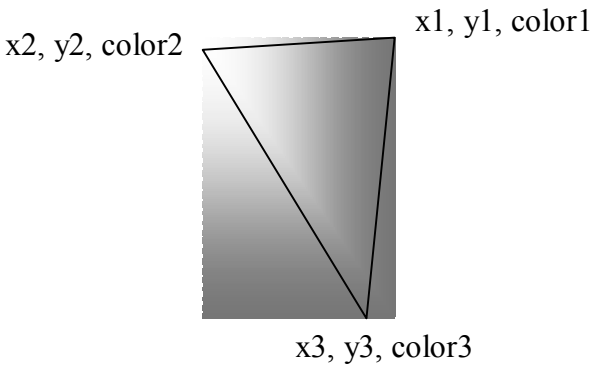


Рис. 4 Билинейная интерполяция по Гуро

Это позволяет достаточно реалистично передавать эффекты освещения, при этом поддерживая приемлемую производительность.

Глава 3.
Состав и структура программы

1.1. Реализация проекта

3.1.1. Структура программы

Программа состоит из двух частей. Одна часть занимается рендерингом примитивов и написана в стиле Direct3D 8.0. Эта часть программы способна компилироваться под тремя различными компиляторами: Borland C++ 3.1, GCC, Microsoft Visual C++ 6.0. Другая часть программы создает интерфейс пользователя. Создание и обработка кнопок в стиле Windows API значительно бы расширило и усложнило программу. Также в программе не используется мышь. Кнопки управляются при помощи клавиатуры. Код их создания различен для каждого компилятора.

3.1.2. Описание эмуляции интерфейса Microsoft Direct3D 8.0

Для эмуляции интерфейса Microsoft Direct3D 8.0 было создано четыре класса: IUnknown, IDirect3D8, IDirect3DDevice8, IDirect3DVertexBuffer8 (рис 3)

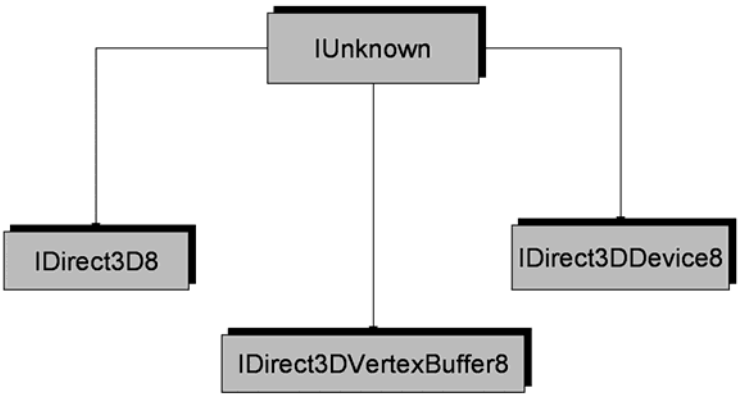


Рис. 5. Наследование классов

3.1.3. Класс IUnknown

Все объекты интерфейсов Direct3D 8.0 создаются при помощи функций, которые требуют указатель на указатель на будущий объект. Внутри этих функций объект создается динамически — при помощи оператора new. Возникла проблема утечки памяти, так как динамически созданный объект необходимо явно удалять при помощи оператора delete. В технологии COM, на основе которой написан Direct3D под Windows для удаления объектов используется функция Release(). В библиотеке эмуляции также присутствует эта функция. Для удаления объектов функция Release() была сделана виртуальной и по умолчанию удаляла текущий объект (delete this).

Функция	*Описание*
Release()	Данная функция используется для удаления текущего объекта. Переопределяется в производных классах так, чтобы объекты удаляли данные дочерних объектов. Обязательно должна вызываться при завершении программы.

3.1.4. Класс IDirect3D8

Данный класс имеет только одну полезную фикцию, которая создает объекты класса IDirect3DDevice8.

Функция	*Описание*
CreateDevice()	Данная функция создает объекты класса IDirect3DDevice8. При завершении программы обязательно должна вызываться его функция Release.

3.1.5. Класс IDirect3DDevice8

Является основным классом библиотеки. При помощи объектов этого класса осуществляется управление процессом рендеринга примитивов и собственно сам процесс рендеринга.

Функция	*Описание*
SetRenderState()	При помощи данной функции имеется возможность контролировать процесс рендеринга. Например, можно включить или отключить освещение, назначить уровень фонового освещения или назначить прорисовку только проволочной модели (прорисовываются только грани, не выполняется заливка треугольников).
CreateVertexBuffer()	Создает буфер вершин (объекты класса IDirect3DVertexBuffer8), данные которого используются в процессе рендеринга. В буфере должна располагаться информация о координатах и нормалях объекта.
Clear()	Данная функция предназначена для очистки содержимого буфера цвета и z-буфера.

Функция	*Описание*
SetMaterial()	Устанавливает текущий материал объекта, т.е. как объект будет взаимодействовать со светом, цвет объекта и т.д.
SetTransform()	Существует три вида преобразования: мировое, камеры и перспективное. Каждый объект, вернее набор его вершин объявляется в своем локальном пространстве. Матрица мирового преобразования (World matrix) служит для преобразования координат объекта из его локального пространства (x1, y1, z1) в мировое (x, y, z) (рис. 4). Матрица преобразования камеры (View matrix) служит для преобразования из мирового пространства (x, y, z) в пространство камеры (x1, y1, z1) (рис. 5). А матрица перспективного преобразования (Projection matrix) служит для перспективной коррекции изображения (с увеличением расстояния от наблюдателя размеры объекта становятся меньше).
DrawPrimitive()	Выполняет рендеринг одного или нескольких треугольников. Использует вершины как списки треугольников (рис. 6) или как вершины цепочки треугольников (рис. 7).
Present()	Данная функция выполняет копирование содержимое буфера цвета в видеопамять, что приводит к обновлению изображения на экране монитора.
SetStreamSource()	Устанавливает буфер вершин в качестве текущего буфера, откуда берет информацию о позиции и норма-

Функция	*Описание*
	лях вершин функция DrawPrimitive.
SetLight()	Устанавливает источник света, с использованием которого вычисляется освещение. Всего может быть до восьми источников света.
LightEnable()	После вызова функции SetLight установлен новый источник света, но его нужно сделать доступным (указать, чтобы он использовался при вычислении освещения).
Release()	Функция очищает содержимое буфера цвета и z-буфера, а также удаляет себя самого (delete this).

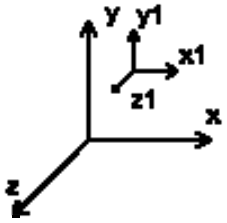


Рис.4. Мировое преобразование координат

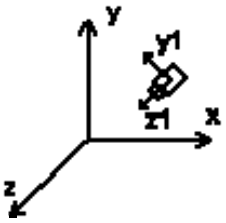


Рис.5. Видовое преобразование координат

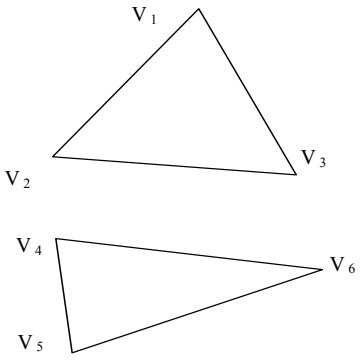


Рис.6. Список треугольников

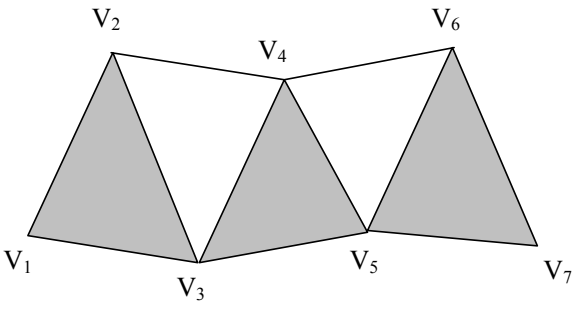


Рис.7. Полоса треугольников

3.1.6. Класс IDirect3DVertexBuffer8

Данный класс представляет собой буфер вершин. В нем хранятся координаты и нормали точек, которые используются при рендеринге.

Функция	*Описание*
Lock()	Функция возвращает указатель на массив внутренних данных, который готов к заполнению координатами и нормалями точек.

3.2. Описание вспомогательных классов

3.2.1. Класс CBackBuffer

Класс представляет собой буфер цвета (буфер, где гранится информация о цветах пикселей), используемый при технологии двойной буферизации.

Функция	*Описание*
Alloc()	Функция заказывает в расширенной памяти буфер, достаточный для хранения всех пикселей экрана монитора.
GetScanLine()	Данная функция возвращает строку пикселей. Такой способ удобен для алгоритма скан конверсии, который обрабатывает изображение по строкам.
SetScanLine()	Данная функция обновляет содержимое скан строки данными из временного буфера. Новые данные получаются после обработки алгоритмом скан конверсии очередной строки.

Функция	*Описание*
Dealloc()	Функция освобождает буфер в расширенной памяти. Вызывается функцией Release класса IDirect3DDevice8.
Clear()	Заполняет содержимое буфера черным цветом.
Flip()	Функция копирует содержимое буфера на экран.

3.2.2. Класс CZBuffer

Во многом аналогичен классу CBackBuffer, но используется для реализации алгоритма *z*-буфера.

Функция	*Описание*
Alloc()	Функция заказывает в расширенной памяти буфер, достаточный для хранения всех z-координат пикселей экрана монитора. Размер каждого элемента равно 2 байта (используется тип short int).
GetScanLine()	Данная функция возвращает z-координаты строки пикселей. Такой способ удобен для алгоритма скан конверсии, который обрабатывает изображение по строкам.
SetScanLine()	Данная функция обновляет содержимое скан строки данными из временного буфера. Новые данные получаются после обработки алгоритмом скан конверсии очередной строки.
Dealloc()	Функция освобождает буфер в расширенной памяти. Вызывается функцией Release класса IDirect3DDevice8.
Clear()	Очищает содержимое максимальной z-координатой (0xffff).

Глава 4.
Инструкция по эксплуатации

1.1. Аппаратные и программные требования

Минимальные аппаратные требования:

Windows-версия:

- Процессор: Pentium 166 MMX
- RAM: Не менее32 Мб
- Установленный DirectX 8.0
- Видеокарта желательна с аппаратным ускорением 3D графики

DOS-версии:

- Процессор: Pentium 133
- RAM: Не менее 4 Мб
- Видеокарта, поддерживающая видеорежим 1024x768x256.

4.1. Описание программы

При запуске программы появляется окно, на котором вращется геометрическая фигура (рис. 8). Внизу расположен ряд кнопок. Кнопки управляются при помощи клавиатуры.

Если мы нажмем клавишу «1», то появится изображение куба (рис. 8).

Клавиша «2» — изображение цилиндра (рис. 9), «3» — сферы.

Если мы нажмем клавишу «4», то появится изображение модели прожектора (рис. 10).

Если мы нажмем клавишу «6», то вместо текущей модели появится ее проволочная модель (рис. 11), «5» — возврат в режим с использованием заливки.

Кнопка «7» или Esc — выход из программы.

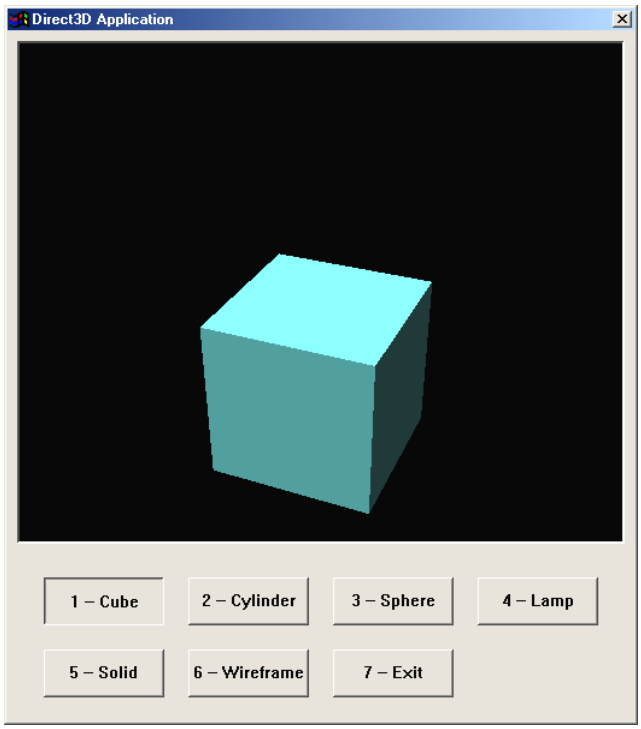


Рис. 8. Модель куба (клавиша «1»)

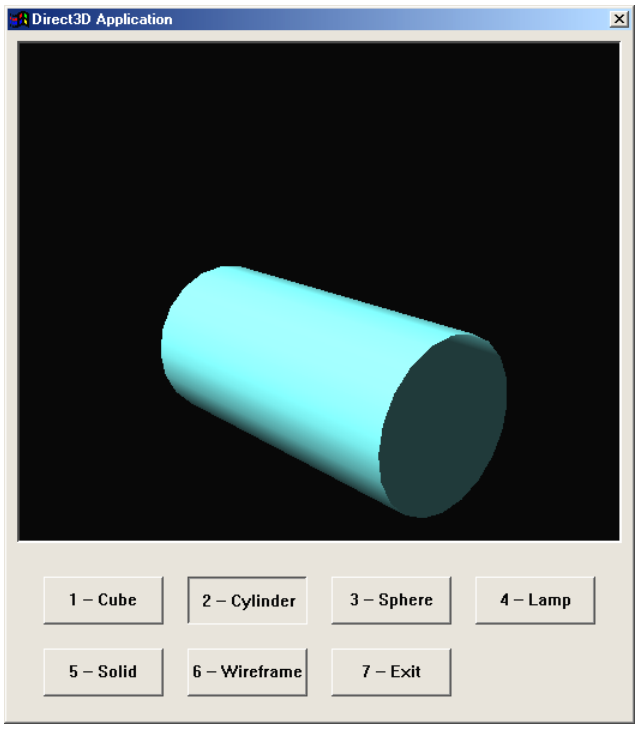


Рис. 9. Модель цилиндра (клавиша «2»)

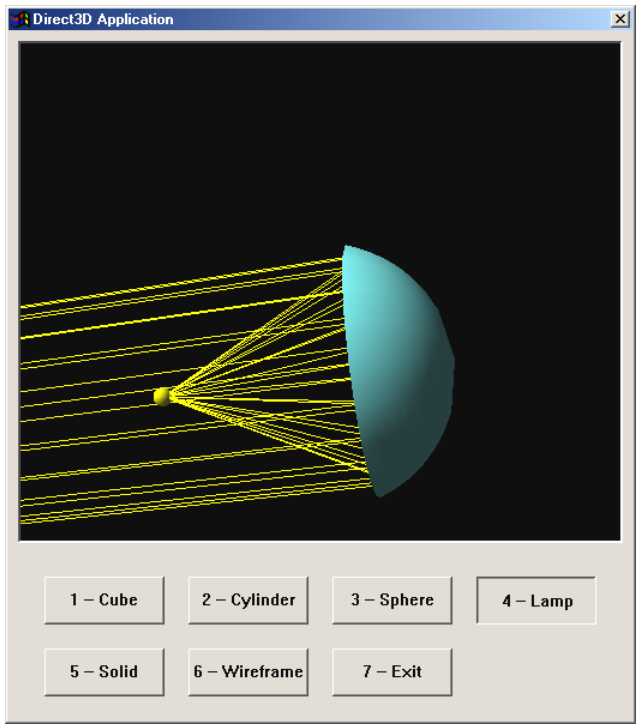


Рис. 10. Модель прожектора (клавиша «4»)

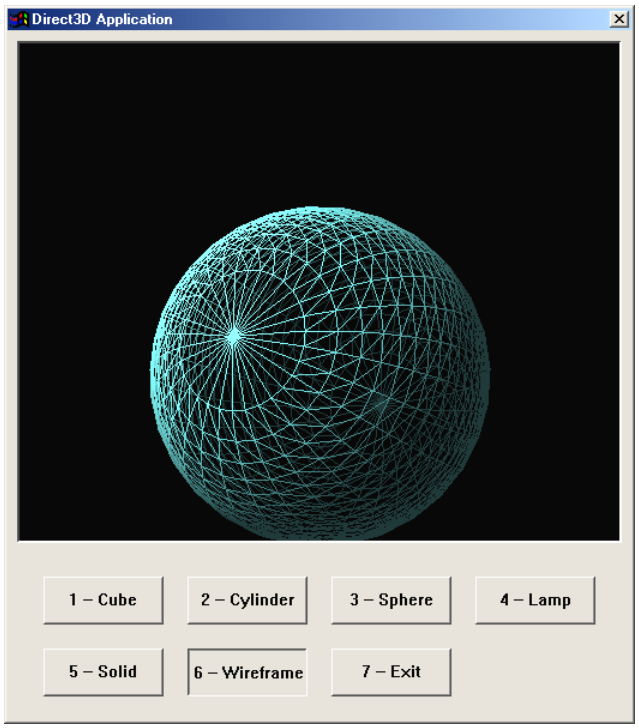


Рис. 11 Проволочная модель (клавиша «6»)

Глава 5.
Заключение

5.1. Достижение совместимости с Windows на уровне исходного кода

Конечно же, полной совместимости со всеми возможностями ОС Windows на уровне исходного кода достигнуть не удалось. Однако такая задача и не ставилась. В работе были эмулированы только те основные возможности, которые необходимы для функционирования простейшего приложения под Windows.

Не предполагалось, что один и тот же EXE файл будет адекватно работать под Windows, и под DOS. Предполагалась возможность компиляции под DOS с использованием компилятора Borland C++ 3.1 и возможность компиляции под Windows с использованием компилятора Microsoft Visual C++ 6.0 и правильное исполнение соответствующих версий программы.

5.2. Использование полноэкранного режима

Год назад, в 8 классе, автором была написана графическая оконная библиотека. Она работала в режиме SVGA, использовала расширенную память и основывалась на стратегии двойной буферизации. Но ее интерфейс (API) ввиду неопытности разработчика был переусложнен, и эмуляция API Windows при помощи оконной библиотеки оказалась чрезвычайно сложной. Для преодоления этого ограничения приложение, работая и под управлением ОС DOS и под управлением ОС Windows, использует полноэкранный режим.

5.3. Выводы

В работе:

- Был создан конвейер рендеринга
- Были эмулированы функции Windows API.
- Был эмулирован интерфейс Microsoft Direct3D 8.0
- Была создана демонстрационная программа
- Была созданы версии библиотеки под различные компиляторы

6. Используемая литература

1. М.Л.Краснов, А.И.Киселев, Г.И.Макаренко, Е.В.Шикин
2. В.И.Заляпин, С.К.Соболев. Вся высшая математика. Том 1. М., 2001.
3. Э. Эйнджел. Программирование на OpenGL. М., 2001.
4. Б. Страуструп. Язык программирования C++. 3-е изд. СПб, 2001.
5. Г. Корн., Т. Корн. Справочник по математике для научных работников и инженеров. М., 1972.

Приложение.
Исходные тексты программ

П.1. Платформно-независимый код

Файл WinMain.cpp

```
#include "winlib.h"
#include <stdio.h>
#include <time.h>
#include "Ctrls.h"

#define SAFE_RELEASE(n) if (n) { n->Release (); n = NULL; }
HWND hWnd;

void OnInit ();
void OnRender ();
void OnDestroy ();

void CreateCylinder (float height, float radius, int stacks, LPDIRECT3DVERTEXBUFFER8 lpVB);

void CreateSphere (float radius, int slices, int stacks, LPDIRECT3DVERTEXBUFFER8* lpVB,
                   LPDIRECT3DINDEXBUFFER8* lpIB);
void CreateHalfSphere (float radius, int slices, int stacks, LPDIRECT3DVERTEXBUFFER8* lpVB,
                       LPDIRECT3DINDEXBUFFER8* lpIB);
void BuildLines (float radius, int slices, int stacks, D3DXVECTOR3* pPoint,
                 LPDIRECT3DVERTEXBUFFER8* ppLines, LPDIRECT3DINDEXBUFFER8* ppIB);

#define RENDER_SPHERE

int CurrentObject = 4;
HINSTANCE hInstance;

BOOL bChanged = FALSE;

int FillMode = D3DFILL_SOLID;

struct MyVertex
{
    D3DXVECTOR3 pos, normal;
    MyVertex (float x, float y, float z, float nx, float ny, float nz) :
        pos (x, y, z), normal (nx, ny, nz) { }
};

struct LineVertex
{
    D3DXVECTOR3 pos;
    DWORD Diffuse;
};

#define D3DFVF_LINEVERTEX (D3DFVF_XYZ | D3DFVF_DIFFUSE)

D3DCOLORVALUE GenColor (float r, float g, float b)
{
    D3DCOLORVALUE res = { r, g, b, 1.0f };
    return res;
};

#define D3DFVF_MYVERTEX (D3DFVF_XYZ | D3DFVF_NORMAL)
```

```
#define SLICES 10
#define STACKS 20

#define LINESL 1
#define LINEST 20

BOOL bPause = FALSE;

LRESULT CALLBACK WndProc (HWND hwnd, UINT nMsg, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hdc;

    switch (nMsg) {
    case WM_DESTROY:
        PostQuitMessage (0);
        OnDestroy ();
        break;
    case WM_KEYDOWN:
        if (wParam == 32)  bPause = !bPause;
        ProcessControls (wParam);
        break;
    case WM_PAINT:
        hdc = BeginPaint (hWnd, &ps);
        OnRender ();
        OnPaint (hdc);
        EndPaint (hWnd, &ps);
        break;
    default:
        return DefWindowProc (hwnd, nMsg, wParam, lParam);
    };
    return 0;
}

INT APIENTRY WinMain (HINSTANCE hInst, HINSTANCE, LPSTR, int)
{
    hInstance = hInst;

    MSG msg;
    WNDCLASS wc;
    ZeroMemory (&wc, sizeof (wc));
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hIcon = LoadIcon (hInst, IDI_APPLICATION);
    wc.hInstance = hInst;
    wc.lpfnWndProc = WndProc;
    wc.lpszClassName = "MAIN WINDOW";
    wc.style = CS_DBLCLKS;

    RegisterClass (&wc);

    hWnd = CreateWindowEx (0, "MAIN WINDOW", "Direct3D Application",
                           WS_POPUP | WS_CAPTION | WS_SYSMENU, 100, 100,
                           526, 600, NULL, NULL, hInst, NULL);

    OnInit ();

    ShowWindow (hWnd, SW_SHOW);
    UpdateWindow (hWnd);
    CreateControls ();

    BOOL bExit = FALSE;
    while (!bExit)
    {
        if (PeekMessage (&msg, NULL, 0, 0, PM_NOREMOVE))
        {
            GetMessage (&msg, NULL, 0, 0);
            TranslateMessage (&msg);
            DispatchMessage (&msg);
            if (msg.message == WM_QUIT)
                bExit = TRUE;
        }
        else
            OnRender ();
    }
    OnDestroy ();
    DeleteControls ();
    return 0;
}
```

```
LPDIRECT3D8 lpD3D = NULL;
LPDIRECT3DDEVICE8 lpDevice = NULL;

LPDIRECT3DVERTEXBUFFER8 lpVB [10] = { NULL };
LPDIRECT3DINDEXBUFFER8  lpIB [10] = { NULL };

void CreateObject (int index)
{
    MyVertex* lpData = NULL;
    float w = 0.5f;
    float h = 0.5f;
    float d = 0.5f;

    switch (index) {
    case 1:
        lpDevice->CreateVertexBuffer (24 * sizeof (MyVertex), 0,
                                      D3DFVF_MYVERTEX, D3DPOOL_DEFAULT, &lpVB [0]);

        lpVB [0]->Lock (0, 0, (BYTE**)&lpData, 0);

        lpData [0] = MyVertex (-w, h, d, 0, 0, 1);
        lpData [1] = MyVertex ( w, h, d, 0, 0, 1);
        lpData [2] = MyVertex (-w,-h, d, 0, 0, 1);
        lpData [3] = MyVertex ( w,-h, d, 0, 0, 1);

        lpData [4] = MyVertex ( w, h, d, 1, 0, 0);
        lpData [5] = MyVertex ( w, h,-d, 1, 0, 0);
        lpData [6] = MyVertex ( w,-h, d, 1, 0, 0);
        lpData [7] = MyVertex ( w,-h,-d, 1, 0, 0);

        lpData [8] = MyVertex ( w, h,-d, 0, 0, -1);
        lpData [9] = MyVertex (-w, h,-d, 0, 0, -1);
        lpData [10]= MyVertex ( w,-h,-d, 0, 0, -1);
        lpData [11]= MyVertex (-w,-h,-d, 0, 0, -1);

        lpData [12]= MyVertex (-w, h, d, -1, 0, 0);
        lpData [13]= MyVertex (-w, h,-d, -1, 0, 0);
        lpData [14]= MyVertex (-w,-h, d, -1, 0, 0);
        lpData [15]= MyVertex (-w,-h,-d, -1, 0, 0);

        lpData [16]= MyVertex (-w, h,-d, 0, 1, 0);
        lpData [17]= MyVertex ( w, h,-d, 0, 1, 0);
        lpData [18]= MyVertex (-w, h, d, 0, 1, 0);
        lpData [19]= MyVertex ( w, h, d, 0, 1, 0);

        lpData [20]= MyVertex (-w,-h, d, 0, -1, 0);
        lpData [21]= MyVertex ( w,-h, d, 0, -1, 0);
        lpData [22]= MyVertex (-w,-h,-d, 0, -1, 0);
        lpData [23]= MyVertex ( w,-h,-d, 0, -1, 0);
        lpVB [0]->Unlock ();
        break;

    case 2:
        lpDevice->CreateVertexBuffer ((STACKS * 2 + 2) * sizeof (MyVertex),
                                      0, D3DFVF_MYVERTEX, D3DPOOL_DEFAULT, &lpVB [0]);
        CreateCylinder (2, 0.5f, STACKS, lpVB [0]);
        break;
    case 3:
        CreateSphere (1.0f, 30, 30, &lpVB [0], &lpIB [0]);
        break;

    case 4:
        CreateHalfSphere (0.8f, SLICES, STACKS, &lpVB [0], &lpIB [0]);
        BuildLines (0.78f, LINESL, LINEST, &D3DXVECTOR3 (0, -1, 0), &lpVB [1], &lpIB [1]);
        CreateSphere (0.06f, 10, 10, &lpVB [2], &lpIB [2]);
        break;
    };
}

D3DXMATRIX matWorld, matWorld2, matView, matProj;

void RenderObject (int index)
{
    int c = 0;
    D3DMATERIAL8 mtrl;
    ZeroMemory (&mtrl, sizeof (mtrl));

    switch (index) {
    case 1:
```

```
        for (c = 0; c < 6; c ++)
            lpDevice->DrawPrimitive (D3DPT_TRIANGLESTRIP, c * 4, 2);
        break;

    case 2:
        lpDevice->DrawPrimitive (D3DPT_TRIANGLESTRIP, 0, STACKS * 2);
        break;

    case 3:
        lpDevice->SetIndices (lpIB [0], 0);
        for (c = 0; c < 29; c ++)
            lpDevice->DrawIndexedPrimitive (D3DPT_TRIANGLESTRIP, 0, 900, c * (30 * 2 + 2), 30 * 2);
        break;

    case 4:
        lpDevice->SetIndices (lpIB [0], 0);
        for (int c = 0; c < SLICES - 1; c ++)
            lpDevice->DrawIndexedPrimitive (D3DPT_TRIANGLESTRIP, c * STACKS,
                                            STACKS * 2, c * (STACKS * 2 + 2), STACKS * 2);

        lpDevice->SetRenderState (D3DRS_LIGHTING, FALSE);
        lpDevice->SetStreamSource (0, lpVB [1], sizeof (LineVertex));
        lpDevice->SetVertexShader (D3DFVF_LINEVERTEX);
        lpDevice->SetIndices (lpIB [1], 0);
        lpDevice->DrawIndexedPrimitive (D3DPT_LINELIST, 0, LINESL * LINEST
                                        +1, 0, LINESL * LINEST);
        lpDevice->DrawPrimitive (D3DPT_LINELIST, LINESL * LINEST + 1,
                                 LINESL * LINEST);

        mtrl.Diffuse = mtrl.Ambient = GenColor (1, 1, 0);
        lpDevice->SetMaterial (&mtrl);

        lpDevice->SetRenderState (D3DRS_LIGHTING, TRUE);
        lpDevice->SetStreamSource (0, lpVB [2], sizeof (MyVertex));
        lpDevice->SetVertexShader (D3DFVF_MYVERTEX);
        lpDevice->SetIndices (lpIB [2], 0);
        lpDevice->SetTransform (D3DTS_WORLD, &matWorld2);

        for (c = 0; c < 9; c ++)
            lpDevice->DrawIndexedPrimitive (D3DPT_TRIANGLESTRIP, c * 10,
                                            10 * 2, c * (10 * 2 + 2), 10 * 2);
        break;
    };
}



void OnInit ()
{
    D3DDISPLAYMODE dm;
    D3DPRESENT_PARAMETERS pp;
    ZeroMemory (&pp, sizeof (pp));
    lpD3D = Direct3DCreate8 (D3D_SDK_VERSION);
    lpD3D->GetAdapterDisplayMode (D3DADAPTER_DEFAULT, &dm);
/*
    pp.BackBufferWidth = 1024;
    pp.BackBufferHeight = 768;
    pp.BackBufferFormat = D3DFMT_A8R8G8B8;
    pp.BackBufferCount = 1;
    pp.MultiSampleType = D3DMULTISAMPLE_NONE;
    pp.SwapEffect = D3DSWAPEFFECT_FLIP;
    pp.hDeviceWindow = hWnd;
    pp.Windowed = FALSE;
    pp.EnableAutoDepthStencil = TRUE;
    pp.AutoDepthStencilFormat = D3DFMT_D16;
    pp.Flags = 0;
    pp.FullScreen_RefreshRateInHz = D3DPRESENT_RATE_DEFAULT;
    pp.FullScreen_PresentationInterval = D3DPRESENT_INTERVAL_DEFAULT;
*/
    pp.Windowed = TRUE;
    pp.EnableAutoDepthStencil = TRUE;
    pp.AutoDepthStencilFormat = D3DFMT_D16;
    pp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    pp.BackBufferFormat = dm.Format;

    if (FAILED(lpD3D->CreateDevice (D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
                                    D3DCREATE_SOFTWARE_VERTEXPROCESSING, &pp, &lpDevice)))
        lpD3D->CreateDevice (D3DADAPTER_DEFAULT, D3DDEVTYPE_REF, hWnd,
                             D3DCREATE_SOFTWARE_VERTEXPROCESSING, &pp, &lpDevice);
```

```
    lpDevice->SetRenderState (D3DRS_CULLMODE, D3DCULL_NONE);
    lpDevice->SetRenderState (D3DRS_ZENABLE, TRUE);

    CreateObject (CurrentObject);
}

void OnRender ()
{
    if (bChanged)
    {
        for (int c = 0; c < 10; c ++)
        {
            SAFE_RELEASE (lpVB [c]);
            SAFE_RELEASE (lpIB [c]);
        }
        CreateObject (CurrentObject);
        bChanged = FALSE;
    }
    lpDevice->Clear (NULL, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, 0, 1, 0);

    lpDevice->BeginScene ();

    RECT rect;
    GetClientRect (hWnd, &rect);

    static float a = 0;
    D3DXMatrixRotationZ (&matWorld, a);

    D3DXMATRIX tmp;
    D3DXMatrixRotationX (&tmp, a / 2);

    matWorld = matWorld * tmp;
    D3DXMatrixIdentity (&matWorld2);
    matWorld2 (3, 1) = -1;
    matWorld2 = matWorld2 * matWorld;

//  if (!bPause)
        a += 0.03f;

    D3DXMatrixLookAtLH (&matView, &D3DXVECTOR3 (0, 0, -5),
                &D3DXVECTOR3 (0, 0, 0), &D3DXVECTOR3 (0, 1, 0));
    D3DXMatrixPerspectiveFovLH (&matProj, D3DX_PI / 4,
                (float)rect.right / rect.bottom, 1, 100);

    lpDevice->SetTransform (D3DTS_WORLD, &matWorld);
    lpDevice->SetTransform (D3DTS_VIEW, &matView);
    lpDevice->SetTransform (D3DTS_PROJECTION, &matProj);

    D3DMATERIAL8 mtrl;
    ZeroMemory (&mtrl, sizeof (mtrl));
    mtrl.Diffuse = mtrl.Ambient = GenColor (0.5, 1, 1);
    mtrl.Power = 100;

    lpDevice->SetMaterial (&mtrl);

    lpDevice->SetRenderState (D3DRS_FILLMODE, FillMode);

    D3DLIGHT8 light;
    ZeroMemory (&light, sizeof light);
    light.Type = D3DLIGHT_DIRECTIONAL;
    light.Direction = D3DXVECTOR3 (1, -1, 1);
    D3DXVec3Normalize ((D3DXVECTOR3*)&light.Direction, (D3DXVECTOR3*)&light.Direction);
    light.Diffuse = GenColor (1, 1, 1);
    lpDevice->SetLight (0, &light);
    lpDevice->LightEnable (0, TRUE);
    lpDevice->SetRenderState (D3DRS_LIGHTING, TRUE);
    lpDevice->SetRenderState (D3DRS_AMBIENT, 0x00303030L);

    lpDevice->SetStreamSource (0, lpVB [0], sizeof (MyVertex));
    lpDevice->SetVertexShader (D3DFVF_MYVERTEX);

    RenderObject (CurrentObject);

    lpDevice->EndScene ();

    rect.top = 10;
    rect.left = 10;
    rect.right -= 10;
```

```
    rect.bottom -= 150;

    lpDevice->Present (&rect, &rect, NULL, NULL);
}

void OnDestroy ()
{
    SAFE_RELEASE (lpD3D);
    SAFE_RELEASE (lpDevice);
    for (int c = 0; c < 10; c ++)
    {
        SAFE_RELEASE (lpVB [c]);
        SAFE_RELEASE (lpIB [c]);
    }
}

void CreateCylinder (float height, float radius, int stacks, LPDIRECT3DVERTEXBUFFER8 lpVB)
{
    MyVertex* lpData = NULL;
    lpVB->Lock (0, 0, (BYTE**)&lpData, 0);

    for (int c = 0; c < stacks; c ++)
    {
        FLOAT angle = (D3DX_PI * 2) / stacks * c;
        float x = (float)sin (angle);
        float z = (float)cos (angle);
        lpData [c * 2 + 0] = MyVertex (x * radius,  height / 2,
                               z * radius, x, 0, z);
        lpData [c * 2 + 1] = MyVertex (x * radius, -height / 2,
                               z * radius, x, 0, z);
    }
    lpData [c * 2 + 0] = lpData [0];
    lpData [c * 2 + 1] = lpData [1];

    lpVB->Unlock ();
}

const float Coef = 1 / 1.5f;

void CreateHalfSphere (float radius, int slices, int stacks, LPDIRECT3DVERTEXBUFFER8* lpVB,
                       LPDIRECT3DINDEXBUFFER8* lpIB)
{
    lpDevice->CreateVertexBuffer (slices * stacks * sizeof (MyVertex), 0,
                      D3DFVF_MYVERTEX, D3DPOOL_DEFAULT, lpVB);
    lpDevice->CreateIndexBuffer ((slices - 1) * (stacks * 2 + 2) *
              sizeof (WORD), 0, D3DFMT_INDEX16, D3DPOOL_DEFAULT, lpIB);

    MyVertex* lpData = NULL;
    (*lpVB)->Lock (0, 0, (BYTE**)&lpData, 0);

    for (int y = 0; y < slices; y ++)
    {
        float ypos = -(radius * Coef / (slices - 1) * y - radius);
        float xpos = (float)sqrt (radius * radius - ypos * ypos);
        for (int x = 0; x < stacks; x ++)
        {
            float angle = D3DX_PI * 2 / stacks * x;
            D3DXMATRIX matRotate;
            D3DXMatrixRotationY (&matRotate, angle);
            D3DXVECTOR3 vec (xpos, ypos, 0);
            D3DXVec3TransformCoord (&vec, &vec, &matRotate);

            lpData [y * stacks + x].pos = vec;
            D3DXVec3Normalize (&vec, &vec);
            lpData [y * stacks + x].normal = vec;
        }
    }

    WORD* lpIndices = NULL;
    (*lpIB)->Lock (0, 0, (BYTE**)&lpIndices, 0);

    for (y = 0; y < slices - 1; y ++)
    {
        for (int x = 0; x < stacks; x ++)
        {
            lpIndices [y * (stacks * 2 + 2) + 2 * x]     = y * stacks + x;
            lpIndices [y * (stacks * 2 + 2) + 2 * x + 1] = (y + 1) * stacks + x;
        }
        lpIndices [y * (stacks * 2 + 2) + 2 * x]   = y * stacks;
```

```
        lpIndices [y * (stacks * 2 + 2) + 2 * x + 1] = (y + 1) * stacks;
    }

    (*lpIB)->Unlock ();
    (*lpVB)->Unlock ();
}

void BuildLines (float radius, int slices, int stacks, D3DXVECTOR3* pPoint,
                 LPDIRECT3DVERTEXBUFFER8* ppLines, LPDIRECT3DINDEXBUFFER8* ppIB)
{
    lpDevice->CreateVertexBuffer
                  ((slices * stacks + 1 + slices * stacks * 2) * sizeof (LineVertex),
                    0, D3DFVF_LINEVERTEX, D3DPOOL_DEFAULT, ppLines);
    lpDevice->CreateIndexBuffer (slices * stacks * 2 * sizeof (WORD),
                                 0, D3DFMT_INDEX16, D3DPOOL_DEFAULT, ppIB);

    LineVertex* lpData = NULL;
    (*ppLines)->Lock (0, 0, (BYTE**)&lpData, 0);

    lpData [0].pos = *pPoint;
    lpData [0].Diffuse = 0x00ffff00L;
    lpData ++;

    for (int y = 0; y < slices; y ++)
    {
//        float ypos = -(radius * Coef / (slices - 1) * y - radius);
        float ypos = -(radius * Coef / (10 - 1) * 7 - radius);
        float xpos = (float)sqrt (radius * radius - ypos * ypos);
        for (int x = 0; x < stacks; x ++)
        {
            float angle = D3DX_PI * 2 / stacks * x;
            D3DXMATRIX matRotate;
            D3DXMatrixRotationY (&matRotate, angle);
            D3DXVECTOR3 vec (xpos, ypos, 0);
            D3DXVec3TransformCoord (&vec, &vec, &matRotate);

            lpData [y * stacks + x].pos = vec;
            lpData [y * stacks + x].Diffuse = 0x00ffff00L;
        }
    }

    int start = slices * stacks;

    for (y = 0; y < slices; y ++)
    {
        float ypos = -(radius * Coef / (10 - 1) * 7 - radius);
        float xpos = (float)sqrt (radius * radius - ypos * ypos);
        for (int x = 0; x < stacks; x ++)
        {
            float angle = D3DX_PI * 2 / stacks * x;
            D3DXMATRIX matRotate;
            D3DXMatrixRotationY (&matRotate, angle);
            D3DXVECTOR3 vec (xpos, ypos, 0);
            D3DXVec3TransformCoord (&vec, &vec, &matRotate);

            lpData [y * stacks * 2 + x * 2 + start].pos = vec;
            lpData [y * stacks * 2 + x * 2 + start].Diffuse = 0x00ffff00L;
            D3DXVECTOR3 vec2 (vec.x, -3, vec.z);
            lpData [y * stacks * 2 + x * 2 + 1 + start].pos = vec2;
            lpData [y * stacks * 2 + x * 2 + 1 + start].Diffuse = 0x00ffff00L;
        }
    }
    (*ppLines)->Unlock ();

    WORD* lpIndices = NULL;
    (*ppIB)->Lock (0, 0, (BYTE**)&lpIndices, 0);

    for (y = 0; y < slices; y ++)
        for (int x = 0; x < stacks; x ++)
        {
            lpIndices [y * stacks * 2 + x * 2] = 0;
            lpIndices [y * stacks * 2 + x * 2 + 1] = y * stacks + x + 1;
        }

    (*ppIB)->Unlock ();
}

void CreateSphere (float radius, int slices, int stacks, LPDIRECT3DVERTEXBUFFER8* lpVB,
                   LPDIRECT3DINDEXBUFFER8* lpIB)
```

```
{
    lpDevice->CreateVertexBuffer (slices * stacks * sizeof (MyVertex), 0,
                                  D3DFVF_MYVERTEX, D3DPOOL_DEFAULT, lpVB);
    lpDevice->CreateIndexBuffer ((slices - 1) * (stacks * 2 + 2) *
                                  sizeof (WORD), 0, D3DFMT_INDEX16, D3DPOOL_DEFAULT, lpIB);

    MyVertex* lpData = NULL;
    (*lpVB)->Lock (0, 0, (BYTE**)&lpData, 0);

    for (int y = 0; y < slices; y ++)
    {
        float ypos = -(radius * 2 / (slices - 1) * y - radius);
        float xpos = (float)sqrt (radius * radius - ypos * ypos);
        for (int x = 0; x < stacks; x ++)
        {
            float angle = D3DX_PI * 2 / stacks * x;
            D3DXMATRIX matRotate;
            D3DXMatrixRotationY (&matRotate, angle);
            D3DXVECTOR3 vec (xpos, ypos, 0);
            D3DXVec3TransformCoord (&vec, &vec, &matRotate);

            lpData [y * stacks + x].pos = vec;
            D3DXVec3Normalize (&vec, &vec);
             lpData [y * stacks + x].normal = vec;
        }
    }

    WORD* lpIndices = NULL;
    (*lpIB)->Lock (0, 0, (BYTE**)&lpIndices, 0);

    for (y = 0; y < slices - 1; y ++)
    {
        for (int x = 0; x < stacks; x ++)
        {
            lpIndices [y * (stacks * 2 + 2) + 2 * x]   = y * stacks + x;
            lpIndices [y * (stacks * 2 + 2) + 2 * x + 1] = (y + 1) * stacks + x;
        }
        lpIndices [y * (stacks * 2 + 2) + 2 * x]   = y * stacks;
        lpIndices [y * (stacks * 2 + 2) + 2 * x + 1] = (y + 1) * stacks;
    }

    (*lpIB)->Unlock ();
    (*lpVB)->Unlock ();
}
```

Файл Ctrls.h

```
#ifndef CONTROLS_H
#define CONTROLS_H

#include "Winlib.h"

void CreateControls ();
void ProcessControls (char ch);
void DeleteControls ();
void OnPaint (HDC hdc);

#endif
```

П.2. Windows-версия библиотеки

Файл WinLib.h

```
#ifndef WINLIB_H
#define WINLIB_H

#include <windows.h>
#include <d3d8.h>
```

```
#include <d3dx8.h>

#endif
```

Файл Ctrls.cpp

```
#include "Indep\Ctrls.h"
#include "WinLib.h"
extern HWND hWnd;

HWND Buttons [7] = { NULL };
extern HINSTANCE hInstance;

extern int CurrentObject;
extern BOOL bChanged;
extern int FillMode;

void FillRect (HDC hdc, int x1, int y1, int x2, int y2, COLORREF color)
{
    HBRUSH br = CreateSolidBrush (color);
    RECT rect = { x1, y1, x2, y2 };
    FillRect (hdc, &rect, br);
    DeleteObject (br);
}

void CreateControls ()
{
    RECT rect;
    GetClientRect (hWnd, &rect);
    rect.top = rect.bottom - 150;

    Buttons [0] = CreateWindow ("BUTTON", "1 -- Cube",
                                WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE, 30,  rect.top + 30, 100, 40,
                                hWnd, (HMENU)100, hInstance, NULL);
    Buttons [1] = CreateWindow ("BUTTON", "2 -- Cylinder",
                                WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE, 150, rect.top + 30, 100, 40,
                                hWnd, (HMENU)100, hInstance, NULL);
    Buttons [2] = CreateWindow ("BUTTON", "3 -- Sphere",
                                WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE, 270, rect.top + 30, 100, 40,
                                hWnd, (HMENU)100, hInstance, NULL);
    Buttons [3] = CreateWindow ("BUTTON", "4 -- Lamp",
                                WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE, 390, rect.top + 30, 100, 40,
                                hWnd, (HMENU)100, hInstance, NULL);
    Buttons [4] = CreateWindow ("BUTTON", "5 -- Solid",
                                WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE, 30,  rect.top + 90, 100, 40,
                                hWnd, (HMENU)100, hInstance, NULL);
    Buttons [5] = CreateWindow ("BUTTON", "6 -- Wireframe",
                                WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE, 150, rect.top + 90, 100, 40,
                                hWnd, (HMENU)100, hInstance, NULL);
    Buttons [6] = CreateWindow ("BUTTON", "7 -- Exit",
                                WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE, 270, rect.top + 90, 100, 40,
                                hWnd, (HMENU)100, hInstance, NULL);

}

int Old = -1;

void ProcessControls (char ch)
{
    int num = ch - '0' - 1;

    if (num == 6)  SendMessage (hWnd, WM_DESTROY, 0, 0);

    SendMessage (Buttons [num], BM_SETSTATE, 1, 0);
    if (Old != -1 && Old != num)  SendMessage (Buttons [Old], BM_SETSTATE, 0, 0);
    if (num >= 0 && num < 4)
    {
        CurrentObject = num + 1;
    }
    if (num >= 4 && num < 6)
    {
        switch (num) {
        case 4:
            FillMode = D3DFILL_SOLID;
            break;
        case 5:
```

```
            FillMode = D3DFILL_WIREFRAME;
            break;
        };
    }

    Old = num;
    SendMessage (Buttons [num], BM_SETSTATE, 1, 0);
    if (Old != -1 && Old != num)  SendMessage (Buttons [Old], BM_SETSTATE, 0, 0);
    bChanged = TRUE;
}

void DeleteControls ()
{
    for (int c = 0; c < 7; c ++)
    {
        DestroyWindow (Buttons [c]);

    }
}

void OnPaint (HDC hdc)
{
    RECT rect;
    GetClientRect (hWnd, &rect);
    FillRect (hdc, 0, 0, 8, rect.bottom, RGB (212, 208, 200));
    FillRect (hdc, rect.right - 8, 0, rect.right, rect.bottom, RGB (212, 208, 200));
    FillRect (hdc, 8, 0, rect.right - 8, 8, RGB (212, 208, 200));
    FillRect (hdc, 8, rect.bottom - 148, rect.right - 8, rect.bottom, RGB (212, 208, 200));

    HPEN hPen = CreatePen (PS_SOLID, 1, RGB (64, 64, 64));
    HGDIOBJ hOldPen = SelectObject (hdc, hPen);

    MoveToEx (hdc, 8, 8, NULL);    LineTo (hdc, rect.right - 8, 8);
    MoveToEx (hdc, 9, 9, NULL);    LineTo (hdc, rect.right - 9, 9);

    MoveToEx (hdc, 8, 8, NULL);    LineTo (hdc, 8, rect.bottom - 148);
    MoveToEx (hdc, 9, 9, NULL);    LineTo (hdc, 9, rect.bottom - 149);

    DeleteObject (hPen);

    hPen = CreatePen (PS_SOLID, 1, RGB (255, 255, 255));
    SelectObject (hdc, hPen);

    MoveToEx (hdc, 8, rect.bottom - 149, NULL);   LineTo (hdc, rect.right - 8, rect.bottom - 149);
    MoveToEx (hdc, 9, rect.bottom - 150, NULL);   LineTo (hdc, rect.right - 9, rect.bottom - 150);

    MoveToEx (hdc, rect.right - 9, 8, NULL);  LineTo (hdc, rect.right - 9, rect.bottom - 149);
    MoveToEx (hdc, rect.right - 10, 9, NULL); LineTo (hdc, rect.right - 10, rect.bottom - 150);

    SelectObject (hdc, hOldPen);
    DeleteObject (hPen);
}
```

П.3. DOS-версия библиотеки

Файл ID3D.h

```
#ifndef IDIRECT3D_H
#define IDIRECT3D_H

#define D3D_SDK_VERSION 0x8

#include "WinLib\WinLib2.h"
#include "Direct3D\IUnknown\IUnknown.h"

enum D3DFORMAT
{
    D3DFMT_A8R8G8B8,
    D3DFMT_INDEX16,
    D3DFMT_D16
};
```

```
struct D3DDISPLAYMODE
{
    UINT Width;
    UINT Height;
    UINT RefreshRate;
    D3DFORMAT Format;
};

enum D3DMULTISAMPLETYPE { D3DMULTISAMPLE_NONE };
enum D3DSWAPEFFECT { D3DSWAPEFFECT_DISCARD };

#define D3DADAPTER_DEFAULT                          0
#define D3DPRESENT_RATE_DEFAULT               0
#define D3DPRESENT_INTERVAL_DEFAULT           0
#define D3DCREATE_SOFTWARE_VERTEXPROCESSING   0

struct D3DPRESENT_PARAMETERS
{
    UINT BackBufferWidth;
    UINT BackBufferHeight;
    D3DFORMAT BackBufferFormat;
    UINT BackBufferCount;

    D3DMULTISAMPLETYPE MultiSampleType;

    D3DSWAPEFFECT SwapEffect;
    HWND hDeviceWindow;
    BOOL Windowed;
    BOOL EnableAutoDepthStencil;
    D3DFORMAT AutoDepthStencilFormat;
    DWORD Flags;

    UINT FullScreen_RefreshRateInHz;
    UINT FullScreen_PresentationInterval;
};

enum D3DDEVTYPE
{
    D3DDEVTYPE_HAL,
    D3DDEVTYPE_REF
};

class IDirect3DDevice8;

typedef class IDirect3D8 : public IUnknown
{
public:
    HRESULT GetAdapterDisplayMode (int adapter, D3DDISPLAYMODE* lpDisplayMode);
    HRESULT CreateDevice (int adapter, D3DDEVTYPE type, HWND hFocusWnd, DWORD BehaviorFlags,
D3DPRESENT_PARAMETERS* pPP, IDirect3DDevice8** ppDevice);
} *LPDIRECT3D8;

LPDIRECT3D8 Direct3DCreate8 (int version);

#endif
```

Файл ID3D.cpp

```
#include "Direct3D\ID3D\ID3D.h"
#include "Direct3D\IDevice\IDevice.h"

#pragma argsused
HRESULT IDirect3D8::GetAdapterDisplayMode (int adapter, D3DDISPLAYMODE* lpDisplayMode)
{
    lpDisplayMode->Format = D3DFMT_A8R8G8B8;
    return 0;
}

#pragma argsused
HRESULT IDirect3D8::CreateDevice (int adapter, D3DDEVTYPE type, HWND hFocusWnd, DWORD BehaviorFlags,
D3DPRESENT_PARAMETERS* pPP, IDirect3DDevice8** ppDevice)
{
    *ppDevice = new IDirect3DDevice8;
    return 0;
```

```
}

#pragma argsused
LPDIRECT3D8 Direct3DCreate8 (int version)
{
    return new IDirect3D8;
}
```

Файл IDevice.h

```
#ifndef IDEVICE_H
#define IDEVICE_H

#include <math.h>

#include "BackBuf\backBuf.h"
#include "ZBuf\ZBuf.h"

#include "Direct3D\ID3D\ID3D.h"
#include "Direct3D\Matrix\Matrix.h"

#define D3DCULL_NONE 0
#define D3DCULL_CCW 1

#define D3DFVF_XYZ     0x0001
#define D3DFVF_NORMAL   0x0002
#define D3DFVF_XYZRHW  0x0004
#define D3DFVF_DIFFUSE 0x0008

#define D3DTS_WORLD0x0001
#define D3DTS_VIEW 0x0002
#define D3DTS_PROJECTION 0x0004

enum D3DRENDERSTATETYPE
{
    D3DRS_ZENABLE,
    D3DRS_CULLMODE,
    D3DRS_FILLMODE,
    D3DRS_AMBIENT,
    D3DRS_LIGHTING,
    D3DRS_RS_SIZE
};

#define D3DFILL_SOLID      0
#define D3DFILL_WIREFRAME   1
#define D3DFILL_POINT      2

struct RGNDATA
{
};

enum D3DPRIMITIVETYPE
{
    D3DPT_TRIANGLELIST,
    D3DPT_TRIANGLESTRIP,
    D3DPT_LINELIST,
    D3DPT_LINESTRIP
};

enum D3DPOOL { D3DPOOL_DEFAULT };

struct D3DRECT
{
    LONG x1, y1, x2, y2;
};

typedef DWORD D3DCOLOR;

struct D3DCOLORVALUE
{
    float r, g, b, a;
};

struct D3DMATERIAL8
{
```

```
    D3DCOLORVALUE Diffuse;
    D3DCOLORVALUE Ambient;
    D3DCOLORVALUE Specular;
    D3DCOLORVALUE Emissive;
    float Power;
};

#define D3DLIGHT_DIRECTIONAL 0x0001

struct D3DLIGHT8
{
    D3DCOLORVALUE Diffuse;
    D3DXVECTOR3 Direction;
    UINT Type;
};

class IDirect3DVertexBuffer8;
class IDirect3DIndexBuffer8;

#define D3DCLEAR_TARGET  0x0001
#define D3DCLEAR_ZBUFFER 0x0002

struct Vertex
{
    D3DXVECTOR3 pos;
    D3DXVECTOR3 normal;
    D3DCOLORVALUE color;
    DWORD Diffuse;
    Vertex () { }
    Vertex (D3DXVECTOR3 p, D3DCOLORVALUE c) : pos (p), color (c) { }
};

struct Line
{
    float x;
    float z;
    float stepz;
    float k;
    D3DCOLORVALUE color;
    D3DCOLORVALUE step;
    void Setup (Vertex v1, Vertex v2)
    {
        if (v2.pos.y - v1.pos.y == 0)  return;
        k = (v2.pos.x - v1.pos.x) / fabs (v2.pos.y - v1.pos.y);
        x = v1.pos.x;
        z = v1.pos.z;
        stepz = (v2.pos.z - v1.pos.z) / fabs (v2.pos.y - v1.pos.y);
        color = v1.color;
        step.r = (v2.color.r - v1.color.r) / fabs (v2.pos.y - v1.pos.y);
        step.g = (v2.color.g - v1.color.g) / fabs (v2.pos.y - v1.pos.y);
        step.b = (v2.color.b - v1.color.b) / fabs (v2.pos.y - v1.pos.y);

    }
    void Evaluate ()
    {
        x += k;
        z += stepz;
        color.r += step.r;
        color.g += step.g;
        color.b += step.b;
    }
};

typedef class IDirect3DDevice8 : public IUnknown
{
protected:
    CBackBuffer m_BackBuffer;
    CZBuffer m_ZBuffer;

    unsigned long m_RenderState [D3DRS_RS_SIZE];

    D3DMATERIAL8 m_mtrl;
    D3DLIGHT8 m_lights [8];
    BOOL m_EnableLight [8];

    D3DXMATRIX m_matWorld, m_matView, m_matProj;

    IDirect3DVertexBuffer8* m_lpVertexBuffer;
    IDirect3DIndexBuffer8* m_lpIndexBuffer;
```

```
    UINT m_stride;

    void* m_pVertices;
    void* m_pIndices;

    void RenderTriangle (Vertex* pVertices);
    void SortVertices (Vertex *v);
    D3DCOLORVALUE CalcColor (int index);
    void ProcessLine (Line line1, Line line2, int y);
    void SetPixel (Line line, BYTE* colorplace, WORD* zplace, int x, int y);

    void Lighting (Vertex* pData);

    unsigned char* m_BackBufferData;
    unsigned short* m_ZBufferData;

    float m_zfar;

    void CreateColorPalette ();
    int ColorGetIndex (float r, float g, float b);
    int ColorGetIndex (D3DCOLOR color);

    void CreateGrayPalette ();
    int GrayGetIndex (float r, float g, float b);
    int GrayGetIndex (D3DCOLOR color);

    void CreatePalette ();
    int GetIndex (float r, float g, float b);
    int GetIndex (D3DCOLOR color);

    Vertex* ProcessVertex (Vertex* pOut, Vertex* pVert);

    BOOL CullFace (Vertex* pVertices);

    void GradientLine (Vertex& v1, Vertex& v2);

    DWORD m_FVF;

    RECT m_OldRect;
    RECT m_NewRect;
public:
    IDirect3DDevice8 ();

    CBackBuffer* BackBuffer () { return &m_BackBuffer; }

    HRESULT SetRenderState (D3DRENDERSTATETYPE state, unsigned long value);
    HRESULT CreateVertexBuffer (int Length, DWORD Usage, DWORD FVF, D3DPOOL Pool,
                                IDirect3DVertexBuffer8** ppVertexBuffer);
    HRESULT CreateIndexBuffer  (int Length, DWORD Usage, D3DFORMAT fmt, D3DPOOL Pool,
                                 IDirect3DIndexBuffer8** ppIndexBuffer);
    HRESULT Clear (DWORD Count, const D3DRECT* pRects, DWORD Flags, D3DCOLOR color, float Z, DWORD Stencil);
    HRESULT BeginScene ();
    HRESULT SetMaterial (D3DMATERIAL8* mtrl);
    HRESULT SetTransform (DWORD Type, D3DXMATRIX* pMatrix);
    HRESULT EndScene ();
    HRESULT Present (RECT* pSourceRect, RECT* pDestRect, HWND hDestroyWindowOverride, RGNDATA*
pDirtyRegion);
    HRESULT SetStreamSource (int stream, IDirect3DVertexBuffer8* lpVB, int stride);
    HRESULT SetVertexShader (int handle);
    HRESULT SetIndices (IDirect3DIndexBuffer8* lpIB, int offset);

    HRESULT DrawPrimitive (D3DPRIMITIVETYPE Type, UINT StartVertex, UINT numPrimitives);
    HRESULT DrawIndexedPrimitive (D3DPRIMITIVETYPE Type, UINT StartVertex, UINT NumVertices,
                                  UINT StartIndex, UINT numPrimitives);

    HRESULT SetLight (int number, D3DLIGHT8* pLight);
    HRESULT LightEnable (int number, BOOL value);

    virtual void Release ();
} *LPDIRECT3DDEVICE8;
#endif
```

Файл IDevice.cpp

```
#include "Direct3D\IDevice\IDevice.h"
#include "Direct3D\IVB\IVB.h"
```

```
#include "Direct3D\IIB\IIB.h"
#include <stdio.h>
#include <dac\dac.h>

//#define COLOR

IDirect3DDevice8::IDirect3DDevice8 () : m_zfar (20)
{
    m_ZBuffer.Alloc ();
    m_BackBuffer.Alloc ();

    ZeroMemory (m_lights, sizeof (m_lights));
    ZeroMemory (m_EnableLight, sizeof (m_EnableLight));
    ZeroMemory (m_RenderState, sizeof (m_RenderState));

    m_RenderState [D3DRS_CULLMODE] = D3DCULL_CCW;
    m_RenderState [D3DRS_FILLMODE] = D3DFILL_SOLID;
    CreatePalette ();

    m_OldRect.left = -1;
    m_NewRect.left = -1;
}

#pragma argsused
HRESULT IDirect3DDevice8::SetRenderState (D3DRENDERSTATETYPE state, unsigned long value)
{
    m_RenderState [state] = value;
    return 0;
}

#pragma argsused
HRESULT IDirect3DDevice8::CreateVertexBuffer (int Length, DWORD Usage, DWORD FVF, D3DPOOL Pool,
IDirect3DVertexBuffer8** ppVertexBuffer)
{
    IDirect3DVertexBuffer8* lpVB = new IDirect3DVertexBuffer8;
    lpVB->Create (Length, FVF);
    *ppVertexBuffer = lpVB;
    return 0;
}

#pragma argsused
HRESULT IDirect3DDevice8::CreateIndexBuffer (int Length, DWORD Usage, D3DFORMAT fmt, D3DPOOL Pool,
IDirect3DIndexBuffer8** ppIndexBuffer)
{
    IDirect3DIndexBuffer8* lpIB = new IDirect3DIndexBuffer8;
    lpIB->Create (Length, fmt);
    *ppIndexBuffer = lpIB;
    return 0;
}

HRESULT IDirect3DDevice8::SetIndices (IDirect3DIndexBuffer8* lpIB, int offset)
{
    m_lpIndexBuffer = lpIB;
    return 0;
}

#pragma argsused
HRESULT IDirect3DDevice8::Clear (DWORD Count, const D3DRECT* pRects, DWORD Flags, D3DCOLOR color,
float Z, DWORD Stencil)
{
    if (Flags & D3DCLEAR_TARGET)   m_BackBuffer.Clear (GetIndex (color));
    if (Flags & D3DCLEAR_ZBUFFER)  m_ZBuffer.Clear ();
    m_NewRect.left = -1;
    return 0;
}

#pragma argsused
HRESULT IDirect3DDevice8::BeginScene ()
{
    return 0;
}

#pragma argsused
HRESULT IDirect3DDevice8::SetMaterial (D3DMATERIAL8* mtrl)
{
    m_mtrl = *mtrl;
    return 0;
}
```

```
#pragma argsused
HRESULT IDirect3DDevice8::SetTransform (DWORD Type, D3DXMATRIX* pMatrix)
{
    switch (Type) {
    case D3DTS_WORLD:
        m_matWorld = *pMatrix;
        break;
    case D3DTS_VIEW:
        m_matView = *pMatrix;
        break;
    case D3DTS_PROJECTION:
        m_matProj = *pMatrix;
        break;
    };
        return 0;
}

#pragma argsused
HRESULT IDirect3DDevice8::DrawPrimitive (D3DPRIMITIVETYPE Type, UINT StartVertex,
                                         UINT numPrimitives)
{
    BOOL bRender = TRUE;

    m_lpVertexBuffer->Lock (0, 0, (BYTE**)&m_pVertices, 0);
    D3DXMATRIX matTransform = m_matWorld * m_matView * m_matProj;
    D3DXMATRIX matNormal = m_matWorld;
    D3DXMatrixInverse (&matNormal, NULL, &matNormal);
    D3DXMatrixTranspose (&matNormal, &matNormal);

    Vertex pData[3];
    Vertex tmp [3];

    for (int c = 0; c < numPrimitives; c ++)
    {
        bRender = TRUE;
        int StructStart;

        switch (Type) {
        case D3DPT_TRIANGLELIST:
            StructStart = StartVertex + c * m_stride * 3;

            pData [0].pos    = *((D3DXVECTOR3*)((char*)m_pVertices +
                       StructStart));
            pData [1].pos    = *((D3DXVECTOR3*)((char*)m_pVertices + m_stride +
                       StructStart));
            pData [2].pos    = *((D3DXVECTOR3*)((char*)m_pVertices + m_stride * 2+
                       StructStart));

            pData [0].normal = *((D3DXVECTOR3*)((char*)m_pVertices + StructStart+
                       sizeof (D3DXVECTOR3)));
            pData [1].normal = *((D3DXVECTOR3*)((char*)m_pVertices + m_stride +
                       StructStart + sizeof (D3DXVECTOR3)));
            pData [2].normal = *((D3DXVECTOR3*)((char*)m_pVertices + m_stride*2 +
                       StructStart + sizeof (D3DXVECTOR3)));

            if (CullFace (pData))  bRender = FALSE;

            ProcessVertex (&pData [0], &pData [0]);
            ProcessVertex (&pData [1], &pData [1]);
            ProcessVertex (&pData [2], &pData [2]);

            tmp     [0] = pData [0];
            tmp     [1] = pData [1];
            tmp     [2] = pData [2];
            if (bRender)       RenderTriangle (tmp);

            break;

        case D3DPT_TRIANGLESTRIP:
            if (c == 0)
            {
                StructStart = StartVertex * m_stride;

                pData [0].pos    = *((D3DXVECTOR3*)((char*)m_pVertices +
                           StructStart));
                pData [1].pos    = *((D3DXVECTOR3*)((char*)m_pVertices +
                           m_stride + StructStart));
                pData [2].pos    = *((D3DXVECTOR3*)((char*)m_pVertices +
                           m_stride * 2 + StructStart));
```

```
                pData [0].normal = *((D3DXVECTOR3*)((char*)m_pVertices +
                        StructStart + sizeof (D3DXVECTOR3)));
                pData [1].normal = *((D3DXVECTOR3*)((char*)m_pVertices +
                        m_stride + StructStart +
                        sizeof (D3DXVECTOR3)));
                pData [2].normal = *((D3DXVECTOR3*)((char*)m_pVertices +
                        m_stride * 2 + StructStart +
                        sizeof (D3DXVECTOR3)));

                if (CullFace (pData))  bRender = FALSE;

                ProcessVertex (&pData [0], &pData [0]);
                ProcessVertex (&pData [1], &pData [1]);
                ProcessVertex (&pData [2], &pData [2]);
            }
            else
            {
                StructStart = (StartVertex + c) * m_stride;

                pData [0] = pData [1];
                pData [1] = pData [2];
                pData [2].pos = *((D3DXVECTOR3*)((char*)m_pVertices +
                         StructStart + m_stride * 2));
                pData [2].normal = *((D3DXVECTOR3*)((char*)m_pVertices +
                        StructStart + m_stride * 2 +
                        sizeof (D3DXVECTOR3)));

                if (CullFace (pData))  bRender = FALSE;

                ProcessVertex (&pData [2], &pData [2]);
            }
            tmp     [0] = pData [0];
            tmp     [1] = pData [1];
            tmp     [2] = pData [2];
            if (bRender)       RenderTriangle (tmp);
            break;

        case D3DPT_LINELIST:
//  for (int c = 0; c < numPrimitives; c ++)
    {
            StructStart = StartVertex * m_stride + c * m_stride * 2;

            pData [0].pos = *((D3DXVECTOR3*)((char*)m_pVertices + StructStart));
            pData [1].pos = *((D3DXVECTOR3*)((char*)m_pVertices + m_stride +
                        StructStart));

            if (m_FVF & D3DFVF_DIFFUSE)
            {
                pData [0].Diffuse = *((DWORD*)((char*)m_pVertices +
                         StructStart + sizeof (D3DXVECTOR3)));
                pData [1].Diffuse = *((DWORD*)((char*)m_pVertices + m_stride +
                         StructStart + sizeof (D3DXVECTOR3)));
            }

            ProcessVertex (&pData [0], &pData [0]);
            ProcessVertex (&pData [1], &pData [1]);

            GradientLine (pData [0], pData [1]);
    }
            break;
        };

/*      pData [0].color.r = 1; pData [0].color.g = 0; pData [0].color.b = 0;
        pData [1].color.r = 0; pData [1].color.g = 1; pData [1].color.b = 0;
        pData [2].color.r = 0; pData [2].color.g = 0; pData [2].color.b = 1;
*/
    }

    m_lpVertexBuffer->Unlock ();
    return 0;
}

#pragma argsused
HRESULT IDirect3DDevice8::DrawIndexedPrimitive (D3DPRIMITIVETYPE Type, UINT StartVertex,
                            UINT NumVertices, UINT StartIndex, UINT numPrimitives)
{
    m_lpVertexBuffer->Lock (0, 0, (BYTE**)&m_pVertices, 0);
    m_lpIndexBuffer->Lock (0, 0, (BYTE**)&m_pIndices, 0);
```

```
WORD* pIndices = (WORD*)m_pIndices;
char* pVertices = (char*)m_pVertices;

pIndices += StartIndex;

D3DXMATRIX matTransform = m_matWorld * m_matView * m_matProj;
D3DXMATRIX matNormal = m_matWorld;
D3DXMatrixInverse (&matNormal, NULL, &matNormal);
D3DXMatrixTranspose (&matNormal, &matNormal);

Vertex pV [3];
int Index [3] = { 0 };
char* Vertices [3] = { 0 };
for (int c = 0; c < numPrimitives; c ++)
{
    BOOL bRender = TRUE;

    D3DPRIMITIVETYPE tmp = Type;
    if (tmp == D3DPT_TRIANGLESTRIP && c == 0)   tmp = D3DPT_TRIANGLELIST;

    Vertex t [3];

    switch (tmp) {
        case D3DPT_TRIANGLELIST:
            Index [0] = pIndices [c * 3];
            Index [1] = pIndices [c * 3 + 1];
            Index [2] = pIndices [c * 3 + 2];
            Vertices [0] = &pVertices [Index [0] * m_stride];
            Vertices [1] = &pVertices [Index [1] * m_stride];
            Vertices [2] = &pVertices [Index [2] * m_stride];

            pV [0].pos = *(D3DXVECTOR3*)Vertices [0];
            pV [1].pos = *(D3DXVECTOR3*)Vertices [1];
            pV [2].pos = *(D3DXVECTOR3*)Vertices [2];

            pV [0].normal = *(D3DXVECTOR3*)(Vertices [0] + sizeof (D3DXVECTOR3));
            pV [1].normal = *(D3DXVECTOR3*)(Vertices [1] + sizeof (D3DXVECTOR3));
            pV [2].normal = *(D3DXVECTOR3*)(Vertices [2] + sizeof (D3DXVECTOR3));

           if (CullFace (pV)) bRender = FALSE;

           ProcessVertex (&pV [0], &pV [0]);
            ProcessVertex (&pV [1], &pV [1]);
           ProcessVertex (&pV [2], &pV [2]);

            t [0] = pV [0];
            t [1] = pV [1];
            t [2] = pV [2];

           if (bRender)       RenderTriangle (t);

            break;
        case D3DPT_TRIANGLESTRIP:
            pV [0] = pV [1];
            pV [1] = pV [2];

            Index [2] = pIndices [2 + c];
            Vertices [2] = &pVertices [Index [2] * m_stride];
            pV [2].pos = *(D3DXVECTOR3*)Vertices [2];
            pV [2].normal = *(D3DXVECTOR3*)(Vertices [2] + sizeof (D3DXVECTOR3));

           ProcessVertex (&pV [2], &pV [2]);

            t [0] = pV [0];
            t [1] = pV [1];
            t [2] = pV [2];

           if (CullFace (pV)) bRender = FALSE;
           if (bRender)       RenderTriangle (t);

            break;
        case D3DPT_LINELIST:

            Index [0] = pIndices [c * 2];
            Index [1] = pIndices [c * 2 + 1];
            Vertices [0] = &pVertices [Index [0] * m_stride];
            Vertices [1] = &pVertices [Index [1] * m_stride];

            pV [0].pos = *(D3DXVECTOR3*)Vertices [0];
```

```
                pV [1].pos = *(D3DXVECTOR3*)Vertices [1];

                pV [0].Diffuse = *(DWORD*)(Vertices [0] + sizeof (D3DXVECTOR3));
                pV [1].Diffuse = *(DWORD*)(Vertices [1] + sizeof (D3DXVECTOR3));

                ProcessVertex (&pV [0], &pV [0]);
                ProcessVertex (&pV [1], &pV [1]);

                GradientLine (pV [0], pV [1]);
                break;
            };
        }
}

#pragma argsused
HRESULT IDirect3DDevice8::EndScene ()
{
    return 0;
}

#pragma argsused
HRESULT IDirect3DDevice8::Present (RECT* pSourceRect, RECT* pDestRect, HWND hDestroyWindowOverride,
                                   RGNDATA* pDirtyRegion)
{
    m_BackBuffer.Flip ();
    m_OldRect = m_NewRect;
    return 0;
}

#pragma argsused
HRESULT IDirect3DDevice8::SetStreamSource (int stream, IDirect3DVertexBuffer8* lpVB, int stride)
{
    m_lpVertexBuffer = lpVB;
    m_stride = stride;
    m_FVF = lpVB->FVF ();
    return 0;
}

#pragma argsused
HRESULT IDirect3DDevice8::SetVertexShader (int handle)
{
    return 0;
}

#pragma argsused
HRESULT IDirect3DDevice8::SetLight (int number, D3DLIGHT8* pLight)
{
    m_lights [number] = *pLight;
    return 0;
}

#pragma argsused
HRESULT IDirect3DDevice8::LightEnable (int number, BOOL value)
{
    m_EnableLight [number] = value;
    return 0;
}

#define TOP    pVertices [0]
#define MIDDLE pVertices [1]
#define BOTTOM pVertices [2]

#define UPDATEBOUNDBOX(vertex) \
    if (m_NewRect.left == -1) \
    { \
        m_NewRect.left = m_NewRect.right = (LONG)vertex.x; \
        m_NewRect.top  = m_NewRect.bottom = (LONG)vertex.y; \
    }   \
    if ((LONG)vertex.x < m_NewRect.left)   m_NewRect.left   = (LONG)vertex.x; \
    if ((LONG)vertex.x > m_NewRect.right)  m_NewRect.right  = (LONG)vertex.x; \
    if ((LONG)vertex.y < m_NewRect.top)    m_NewRect.top    = (LONG)vertex.y; \
    if ((LONG)vertex.y > m_NewRect.bottom) m_NewRect.bottom = (LONG)vertex.y; \

void IDirect3DDevice8::RenderTriangle (Vertex* pVertices)
{
    TOP.pos.y -= (TOP.pos.y - (int)TOP.pos.y);
    MIDDLE.pos.y -= (MIDDLE.pos.y - (int)MIDDLE.pos.y);
    BOTTOM.pos.y -= (BOTTOM.pos.y - (int)BOTTOM.pos.y);
    TOP.pos.x -= (TOP.pos.x - (int)TOP.pos.x);
```

```
    MIDDLE.pos.x -= (MIDDLE.pos.x - (int)MIDDLE.pos.x);
    BOTTOM.pos.x -= (BOTTOM.pos.x - (int)BOTTOM.pos.x);

    UPDATEBOUNDBOX (TOP.pos);
    UPDATEBOUNDBOX (MIDDLE.pos);
    UPDATEBOUNDBOX (BOTTOM.pos);

    if (m_RenderState [D3DRS_FILLMODE] == D3DFILL_WIREFRAME)
    {
        GradientLine (pVertices [0], pVertices [1]);
        GradientLine (pVertices [0], pVertices [2]);
        GradientLine (pVertices [1], pVertices [2]);
        return;
    }
    SortVertices (pVertices);

    if (TOP.pos.y == BOTTOM.pos.y) return;

    if (TOP.pos.y != MIDDLE.pos.y)
    {
        Line line1, line2, line3;
        line1.Setup (TOP, MIDDLE);
        line2.Setup (TOP, BOTTOM);
        if (MIDDLE.pos.y != BOTTOM.pos.y)  line3.Setup (MIDDLE, BOTTOM);

        for (int y = (int)TOP.pos.y; y < (int)BOTTOM.pos.y; y ++)
        {
            ProcessLine (line1, line2, y);
            if (y == (int)MIDDLE.pos.y)
                line1 = line3;
            line1.Evaluate ();
            line2.Evaluate ();
        }
    }
    else
    {
        Line line1, line2;
        line1.Setup (BOTTOM, MIDDLE);
        line2.Setup (BOTTOM, TOP);

        line1.Evaluate ();
        line2.Evaluate ();

        for (int y = (int)BOTTOM.pos.y - 1; y >= (int)TOP.pos.y; y --)
        {
            ProcessLine (line1, line2, y);
            line1.Evaluate ();
            line2.Evaluate ();
        }
    }
}

void IDirect3DDevice8::SortVertices (Vertex *v)
{
    for (int c = 0; c < 3; c ++)
        for (int i = c + 1; i < 3; i ++)
            if (v [i].pos.y < v [c].pos.y)
            {
                Vertex tmp = v [c];
                v [c] = v [i];
                v [i] = tmp;
            }
}

D3DCOLORVALUE IDirect3DDevice8::CalcColor (int index)
{
    if (m_lpVertexBuffer->FVF () & D3DFVF_DIFFUSE)
    {
        int offset = 0;
        if (m_lpVertexBuffer->FVF () & D3DFVF_XYZ) offset += sizeof (D3DXVECTOR3);
        else if (m_lpVertexBuffer->FVF () & D3DFVF_XYZRHW) offset += sizeof (D3DXVECTOR3) +
                                                                     sizeof (float);

        DWORD color = *((DWORD*)((BYTE*)m_pVertices + m_stride * index + offset));
        D3DCOLORVALUE res;
        res.b = ((color & 0x000000ffL)     ) / 255.0f;
        res.g = ((color & 0x0000ff00L) >> 8   ) / 255.0f;
        res.r = ((color & 0x00ff0000L) >> 16) / 255.0f;
        res.a = ((color & 0xff000000L) >> 24) / 255.0f;
```

```
        return res;
    }
    else
    {
        D3DCOLORVALUE res;
        ZeroMemory (&res, sizeof (res));
        res.r = 1.0f;
        return res;
    }
}

void IDirect3DDevice8::ProcessLine (Line line1, Line line2, int y)
{
    if (y < 0 || y > 768)  return;
    if (line1.x == line2.x)return;

    if (line1.x > line2.x)
    {
        Line tmp = line1;
        line1 = line2;
        line2 = tmp;
    }

    Line tmp = line2;

    tmp.x = line1.x;
    tmp.z = line1.z;
    tmp.stepz = (line2.z - line1.z) / (line2.x - line1.x);
    tmp.k = 1;
    tmp.color = line1.color;
    tmp.step.r = (line2.color.r - line1.color.r) / (line2.x - line1.x);
    tmp.step.g = (line2.color.g - line1.color.g) / (line2.x - line1.x);
    tmp.step.b = (line2.color.b - line1.color.b) / (line2.x - line1.x);

    m_BackBuffer.GetScanLine (&m_BackBufferData, y);
    m_ZBuffer.GetScanLine (&m_ZBufferData, y);

    int x1 = (int)ceil (line1.x);
    int x2 = (int)ceil (line2.x);
//    if (x2 == line2.x)   x2 --;

    for (int x = x1; x <= x2; x ++)
    {
        if (x >= 0 && x < 1024)
        {
            SetPixel (tmp, m_BackBufferData, m_ZBufferData, x, y);
//            ((DWORD*)m_BackBufferData) [x] = 0x0000003f;
        }
        tmp.Evaluate ();
    }
    m_BackBuffer.SetScanLine (&m_BackBufferData, y);
    m_ZBuffer.SetScanLine (&m_ZBufferData, y);

}

void IDirect3DDevice8::Lighting (Vertex* pData)
{
    if (m_FVF & D3DFVF_NORMAL)
    {
        D3DCOLORVALUE Ambient;
        Ambient.r = ((m_RenderState [D3DRS_AMBIENT] & 0x000000ffL) >> 0) / 255.0f;
        Ambient.g = ((m_RenderState [D3DRS_AMBIENT] & 0x0000ff00L) >> 8) / 255.0f;
        Ambient.b = ((m_RenderState [D3DRS_AMBIENT] & 0x00ff0000L)>> 16) / 255.0f;

        Ambient.r *= m_mtrl.Ambient.r;
        Ambient.g *= m_mtrl.Ambient.g;
        Ambient.b *= m_mtrl.Ambient.b;

        D3DXVECTOR3 Incedence = -m_lights [0].Direction;
        float power = pData->normal * Incedence;
        if (power < 0)  power = 0;
        pData->color.r = m_mtrl.Diffuse.r * power + Ambient.r;
        pData->color.g = m_mtrl.Diffuse.g * power + Ambient.g;
        pData->color.b = m_mtrl.Diffuse.b * power + Ambient.b;

        if (pData->color.r > 1)pData->color.r = 1;
        else if (pData->color.r < 0)   pData->color.r = 0;
```

```
        if (pData->color.g > 1)pData->color.g = 1;
        else if (pData->color.g < 0)   pData->color.g = 0;

        if (pData->color.b > 1)pData->color.b = 1;
        else if (pData->color.b < 0)   pData->color.b = 0;
    }
}

void IDirect3DDevice8::CreateColorPalette ()
{
    for (int r = 0; r < 4; r ++)
    {
        for (int g = 0; g < 8; g ++)
        {
            for (int b = 0; b < 8; b ++)
            {
                int cr = r * 13;
                int cg = g * 9;
                int cb = b * 9;
                if (cr > 63)   cr = 63;
                if (cg > 63)   cg = 63;
                if (cb > 63)   cb = 63;
#ifdef BGI
                setrgbpalette ((r << 6) | (g << 3) | (b), cr, cg, cb);
#else
                SetDacReg ((r << 6) | (g << 3) | (b), cr, cg, cb);
#endif
            }
        }
    }
}

int IDirect3DDevice8::ColorGetIndex (float r, float g, float b)
{
    int ir = (int)((r * 52) / 13);
    int ig = (int)((g * 52) / 9);
    int ib = (int)((b * 52) / 9);
    return (ir << 6) | (ig << 3) | (ib);
}

int IDirect3DDevice8::ColorGetIndex (D3DCOLOR color)
{
    char cr = (char)(color  & 0x000000ffL);
    char cg = (char)((color & 0x0000ff00L) >> 8);
    char cb = (char)((color & 0x00ff0000L) >> 16);

    int ir = (int)(cr / 36.5);
    int ig = (int)(cg / 17.0);
    int ib = (int)(cb / 17.0);
    return (ir << 6) | (ig << 3) | (ib);
}

void IDirect3DDevice8::CreateGrayPalette ()
{
    for (int c = 16; c < 80; c ++)
#ifndef BGI
        SetDacReg (c, (c - 16) / 2, c - 16, c - 16);
#else
        setrgbpalette (c, (c - 16) / 2, (c - 16), (c - 16));
#endif

    SetDacReg (80, 32, 32, 0);

    for (c = 81; c < 145; c ++)
        SetDacReg (c, (c - 81), (c - 81), 0);

}

int IDirect3DDevice8::GrayGetIndex (float r, float g, float b)
{
    if (r == 1 && g == 1 && b == 0)    return 80;
    if (r != 0 && g != 0 && b == 0)
    {
        int res = (int)(r * 63) + 81;
        if (res > 145)     res = 145;
        return res;
    }

    int res = (int)(b * 63);
```

```
    if (res > 63)     res = 63;
    return res + 16;
}

int IDirect3DDevice8::GrayGetIndex (D3DCOLOR color)
{
    if (color == 0x00ffff00L)  return 80;
    char cr = (char)(color  & 0x000000ffL);
    char cg = (char)((color & 0x0000ff00L) >> 8);
    char cb = (char)((color & 0x00ff0000L) >> 16);

    int res = (cr + cg + cb) / 12;
    if (res > 63) res = 63;
    return res;
}

void IDirect3DDevice8::CreatePalette ()
{
#ifdef COLOR
    CreateColorPalette ();
#else
    CreateGrayPalette ();
#endif
}

int IDirect3DDevice8::GetIndex (float r, float g, float b)
{
#ifdef COLOR
    return ColorGetIndex (r, g, b);
#else
    return GrayGetIndex (r, g, b);
#endif
}

int IDirect3DDevice8::GetIndex (D3DCOLOR color)
{
#ifdef COLOR
    return ColorGetIndex (color);
#else
    return GrayGetIndex (color);
#endif
}

Vertex* IDirect3DDevice8::ProcessVertex (Vertex* pOut, Vertex* pVert)
{
    D3DXVec3TransformCoord (&pOut->pos, &pVert->pos, &m_matWorld);

    if (m_FVF & D3DFVF_NORMAL)
    {
        D3DXMATRIX tmp;
        D3DXMatrixInverse (&tmp, NULL, &m_matWorld);
        D3DXMatrixTranspose (&tmp, &tmp);
        D3DXVec3TransformCoord (&pOut->normal, &pVert->normal, &tmp);
        D3DXVec3Normalize (&pOut->normal, &pOut->normal);
        Lighting (pOut);
    }
    else if (m_FVF & D3DFVF_DIFFUSE)
    {
        pOut->Diffuse = pVert->Diffuse;
        pOut->color.r = ((pVert->Diffuse & 0x00ff0000L) >> 16) / 255.0f;
        pOut->color.g = ((pVert->Diffuse & 0x0000ff00L) >>  8) / 255.0f;
        pOut->color.b = ((pVert->Diffuse & 0x000000ffL) >>  0) / 255.0f;

        pOut->color.r = 1;
        pOut->color.g = 1;
        pOut->color.b = 0;
    }

    D3DXMATRIX matTmp = m_matView * m_matProj;
    D3DXVec3TransformCoord (&pOut->pos, &pOut->pos, &matTmp);

    pOut->pos.x = ( pOut->pos.x + 1) * 512;
    pOut->pos.y = (-pOut->pos.y + 1) * 384;

    pOut->pos.z /= m_zfar;
    return pOut;
}

BOOL IDirect3DDevice8::CullFace (Vertex* pVertices)
```

```
{
    if (m_RenderState [D3DRS_CULLMODE] == D3DCULL_NONE)   return FALSE;

    D3DXVECTOR3 tmp [3];
    D3DXMATRIX t = m_matWorld;
    D3DXVec3TransformCoord (&tmp [0], &pVertices [0].pos, &t);
    D3DXVec3TransformCoord (&tmp [1], &pVertices [1].pos, &t);
    D3DXVec3TransformCoord (&tmp [2], &pVertices [2].pos, &t);

    D3DXVECTOR3 v1 = tmp [2] - tmp [0];
    D3DXVECTOR3 v2 = tmp [1] - tmp [0];
    D3DXVECTOR3 Normal;
    D3DXVec3Cross (&Normal, &v1, &v2);
    D3DXVec3Normalize (&Normal, &Normal);

    if (Normal * D3DXVECTOR3 (0, 0, -1) < 0)  return TRUE;

    return FALSE;
}

void IDirect3DDevice8::GradientLine (Vertex& v1, Vertex& v2)
{
    UPDATEBOUNDBOX(v1.pos);
    UPDATEBOUNDBOX(v2.pos);

    int dx = (int)(v2.pos.x - v1.pos.x);
    int dy = (int)(v2.pos.y - v1.pos.y);
    int incx = dx > 0 ? 1 : dx == 0 ? 0 : -1;
    int incy = dy > 0 ? 1 : dy == 0 ? 0 : -1;

    dx = abs (dx);
    dy = abs (dy);
    int p = dx > dy ? dx : dy;

    int x = (int)v1.pos.x;
    int y = (int)v1.pos.y;

    int errx = 0, erry = 0;

    D3DCOLORVALUE color = v1.color;
    D3DCOLORVALUE step;
    step.r = (v2.color.r - v1.color.r) / (p + 2);
    step.g = (v2.color.g - v1.color.g) / (p + 2);
    step.b = (v2.color.b - v1.color.b) / (p + 2);

    float z = v1.pos.z;
    float stepz = (v2.pos.z - v1.pos.z) / (p + 2);

    for (int t = 0; t < p + 2; t ++)
    {
        if ((UINT)m_ZBuffer.GetPixel (x, y) > (UINT)(z * 65535U))
        {
            m_BackBuffer.SetPixel (x, y, GetIndex (color.r, color.g, color.b));
            m_ZBuffer.SetPixel (x, y, (UINT)(z * 65535L));
        }

        errx += dx;
        erry += dy;

        color.r += step.r;
        color.g += step.g;
        color.b += step.b;

        z += stepz;

        if (errx >= p)
        {
            errx -= p;
            x += incx;
        }
        if (erry >= p)
        {
            erry -= p;
            y += incy;
        }
    }
}

void IDirect3DDevice8::Release ()
```

```
{
    m_BackBuffer.Dealloc ();
    m_ZBuffer.Dealloc ();
    IUnknown::Release ();
}
```

Файл IIB.h

```
#ifndef IDIRECT3DINDEXBUFFER8_H
#define IDIRECT3DINDEXBUFFER8_H

#include "Direct3D\IDevice\IDevice.h"
#include "WinLib\WinLib2.h"

typedef class IDirect3DIndexBuffer8 : public IUnknown
{
    char* m_data;
    int m_size;
    D3DFORMAT m_Fmt;
public:
    HRESULT Lock (UINT OffsetToLock, UINT SizeToLock, BYTE** ppbData, DWORD Flags);
    HRESULT Unlock ();

    void Create (int size, D3DFORMAT Fmt);
    virtual void Release ();

    int Size () { return m_size; }
    int FMT () { return m_Fmt; }
} *LPDIRECT3DINDEXBUFFER8;


#endif
```

Файл IIB.cpp

```
#include <Direct3D\IIB\IIB.h>

HRESULT IDirect3DIndexBuffer8::Lock (UINT OffsetToLock, UINT SizeToLock, BYTE** ppbData, DWORD Flags)
{
    *ppbData = (BYTE*)m_data;
}

HRESULT IDirect3DIndexBuffer8::Unlock ()
{

}

void IDirect3DIndexBuffer8::Create (int size, D3DFORMAT Fmt)
{
    m_Fmt = Fmt;
    m_data = new char [size];
}

void IDirect3DIndexBuffer8::Release ()
{
    delete[] m_data;
    IUnknown::Release ();
}
```

Файл IUnknown.h

```
#ifndef IUNKNOWN_H
#define IUNKNOWN_H

class IUnknown
{
public:
    virtual void Release () { delete this; }
};
```

```
#endif
```

Файл IVB.h

```
#ifndef IDIRECT3DVERTEXBUFFER8_H
#define IDIRECT3DVERTEXBUFFER8_H

//#include "Direct3D\IDevice\IDevice.h"
#include "Direct3D\IUnknown\IUnknown.h"
#include "WinLib\WinLib2.h"

typedef unsigned char BYTE;

typedef class IDirect3DVertexBuffer8 : public IUnknown
{
    char* m_data;
    int m_size;
    DWORD m_FVF;
public:
    HRESULT Lock (UINT OffsetToLock, UINT SizeToLock, BYTE** ppbData, DWORD Flags);
    HRESULT Unlock ();

    void Create (int size, DWORD FVF);
    virtual void Release ();

    int Size () { return m_size; }
    DWORD FVF () { return m_FVF; }
} *LPDIRECT3DVERTEXBUFFER8;


#endif
```

Файл IVB.cpp

```
#include "Direct3d\IVB\IVB.h"

#pragma argsused
HRESULT IDirect3DVertexBuffer8::Lock (UINT OffsetToLock, UINT SizeToLock, BYTE** ppbData, DWORD Flags)
{
    *ppbData = (BYTE*)m_data;
    return 0;
}

#pragma argsused
HRESULT IDirect3DVertexBuffer8::Unlock ()
{
    return 0;
}

void IDirect3DVertexBuffer8::Create (int size, DWORD FVF)
{
    m_data = new char [size];
    m_size = size;
    m_FVF = FVF;
}

void IDirect3DVertexBuffer8::Release ()
{
    delete[] m_data;
    IUnknown::Release ();
}
```

Файл Matrix.h

```
#ifndef MATRIX_H
#define MATRIX_H

#include "Direct3D\Vector\Vector.h"
```

```
class D3DXVECTOR3;

struct D3DXMATRIX
{
    union {
        struct {
            float _m11, _m12, _m13, _m14;
            float _m21, _m22, _m23, _m24;
            float _m31, _m32, _m33, _m34;
            float _m41, _m42, _m43, _m44;
        };
        float _m [4][4];
    };
    D3DXMATRIX (float m11, float m12, float m13, float m14,
                float m21, float m22, float m23, float m24,
                float m31, float m32, float m33, float m34,
                float m41, float m42, float m43, float m44)
                {
                    _m [0][0] = m11;    _m [0][1] = m12;
                    _m [0][2] = m13;    _m [0][3] = m14;
                    _m [1][0] = m21;    _m [1][1] = m22;
                    _m [1][2] = m23;    _m [1][3] = m24;
                    _m [2][0] = m31;    _m [2][1] = m32;
                    _m [2][2] = m33;    _m [2][3] = m34;
                    _m [3][0] = m41;    _m [3][1] = m42;
                    _m [3][2] = m43;    _m [3][3] = m44;
                }
    D3DXMATRIX ()
    {
    _m [0][0] = 1;  _m [0][1] = 0; _m [0][2] = 0;  _m [0][3] = 0;
    _m [1][0] = 0;  _m [1][1] = 1; _m [1][2] = 0;  _m [1][3] = 0;
    _m [2][0] = 0;  _m [2][1] = 0; _m [2][2] = 1;  _m [2][3] = 0;
    _m [3][0] = 0;  _m [3][1] = 0; _m [3][2] = 0;  _m [3][3] = 1;
    }

    D3DXMATRIX operator* (const D3DXMATRIX& m)
    {
        return D3DXMATRIX (
        _m [0][0] * m._m [0][0] + _m [0][1] * m._m [1][0] + _m [0][2] * m._m [2][0] + _m [0][3] * m._m [3][0],
        _m [0][0] * m._m [0][1] + _m [0][1] * m._m [1][1] + _m [0][2] * m._m [2][1] + _m [0][3] * m._m [3][1],
        _m [0][0] * m._m [0][2] + _m [0][1] * m._m [1][2] + _m [0][2] * m._m [2][2] + _m [0][3] * m._m [3][2],
        _m [0][0] * m._m [0][3] + _m [0][1] * m._m [1][3] + _m [0][2] * m._m [2][3] + _m [0][3] * m._m [3][3],

        _m [1][0] * m._m [0][0] + _m [1][1] * m._m [1][0] + _m [1][2] * m._m [2][0] + _m [1][3] * m._m [3][0],
        _m [1][0] * m._m [0][1] + _m [1][1] * m._m [1][1] + _m [1][2] * m._m [2][1] + _m [1][3] * m._m [3][1],
        _m [1][0] * m._m [0][2] + _m [1][1] * m._m [1][2] + _m [1][2] * m._m [2][2] + _m [1][3] * m._m [3][2],
        _m [1][0] * m._m [0][3] + _m [1][1] * m._m [1][3] + _m [1][2] * m._m [2][3] + _m [1][3] * m._m [3][3],

        _m [2][0] * m._m [0][0] + _m [2][1] * m._m [1][0] + _m [2][2] * m._m [2][0] + _m [2][3] * m._m [3][0],
        _m [2][0] * m._m [0][1] + _m [2][1] * m._m [1][1] + _m [2][2] * m._m [2][1] + _m [2][3] * m._m [3][1],
        _m [2][0] * m._m [0][2] + _m [2][1] * m._m [1][2] + _m [2][2] * m._m [2][2] + _m [2][3] * m._m [3][2],
        _m [2][0] * m._m [0][3] + _m [2][1] * m._m [1][3] + _m [2][2] * m._m [2][3] + _m [2][3] * m._m [3][3],

        _m [3][0] * m._m [0][0] + _m [3][1] * m._m [1][0] + _m [3][2] * m._m [2][0] + _m [3][3] * m._m [3][0],
        _m [3][0] * m._m [0][1] + _m [3][1] * m._m [1][1] + _m [3][2] * m._m [2][1] + _m [3][3] * m._m [3][1],
        _m [3][0] * m._m [0][2] + _m [3][1] * m._m [1][2] + _m [3][2] * m._m [2][2] + _m [3][3] * m._m [3][2],
        _m [3][0] * m._m [0][3] + _m [3][1] * m._m [1][3] + _m [3][2] * m._m [2][3] + _m [3][3] * m._m [3][3]);
    }
    float& operator() (int col, int row) { return _m [col][row]; }
};

#define D3DX_PI 3.141592654f

D3DXMATRIX* D3DXMatrixRotationX (D3DXMATRIX* pOut, float angle);
D3DXMATRIX* D3DXMatrixRotationY (D3DXMATRIX* pOut, float angle);
D3DXMATRIX* D3DXMatrixRotationZ (D3DXMATRIX* pOut, float angle);
D3DXMATRIX* D3DXMatrixTranslation (D3DXMATRIX* pOut, float dx, float dy, float dz);
D3DXMATRIX* D3DXMatrixIdentity (D3DXMATRIX* pOut);

D3DXMATRIX* D3DXMatrixLookAtLH (D3DXMATRIX* pOut, D3DXVECTOR3* pos, D3DXVECTOR3* target,
                                D3DXVECTOR3* y);
D3DXMATRIX* D3DXMatrixPerspectiveFovLH (D3DXMATRIX* pOut, float angle, float aspect, float znear,
                                        float zfar);
D3DXMATRIX* D3DXMatrixTranspose (D3DXMATRIX* pOut, D3DXMATRIX* pMat);
D3DXMATRIX* D3DXMatrixInverse (D3DXMATRIX* pOut, float* pDet, D3DXMATRIX* pMat);

#endif
```

Файл Matrix.cpp

```
#include "Direct3D\matrix\matrix.h"
#include <math.h>
#include <assert.h>
#include <stdio.h>

#pragma argsused
D3DXMATRIX* D3DXMatrixRotationX (D3DXMATRIX* pOut, float angle)
{
    D3DXMatrixIdentity (pOut);
    float cosangle = cos (angle);
    float sinangle = sin (angle);

    pOut->_m [1][1] = cosangle;
    pOut->_m [1][2] = sinangle;
    pOut->_m [2][1] =-sinangle;
    pOut->_m [2][2] = cosangle;

    return pOut;
}

#pragma argsused
D3DXMATRIX* D3DXMatrixRotationY (D3DXMATRIX* pOut, float angle)
{
    D3DXMatrixIdentity (pOut);
    pOut->_m [0][0] = cos (angle);
    pOut->_m [0][2] = -sin (angle);
    pOut->_m [2][0] = sin (angle);
    pOut->_m [2][2] = cos (angle);
    return pOut;
}

#pragma argsused
D3DXMATRIX* D3DXMatrixRotationZ (D3DXMATRIX* pOut, float angle)
{
    D3DXMatrixIdentity (pOut);
    pOut->_m [0][0] = cos (angle);
    pOut->_m [0][1] = sin (angle);
    pOut->_m [1][0] = -sin (angle);
    pOut->_m [1][1] = cos (angle);
    return pOut;
}

#pragma argsused
D3DXMATRIX* D3DXMatrixTranslation (D3DXMATRIX* pOut, float dx, float dy, float dz)
{
    D3DXMatrixIdentity (pOut);
    pOut->_m [3][0] = dx;
    pOut->_m [3][1] = dy;
    pOut->_m [3][2] = dz;
    return pOut;
}

#pragma argsused
D3DXMATRIX* D3DXMatrixIdentity (D3DXMATRIX* pOut)
{
    pOut->_m [0][0] = 1; pOut->_m [0][1] = 0;
    pOut->_m [0][2] = 0; pOut->_m [0][3] = 0;
    pOut->_m [1][0] = 0; pOut->_m [1][1] = 1;
    pOut->_m [1][2] = 0; pOut->_m [1][3] = 0;
    pOut->_m [2][0] = 0; pOut->_m [2][1] = 0;
    pOut->_m [2][2] = 1; pOut->_m [2][3] = 0;
    pOut->_m [3][0] = 0; pOut->_m [3][1] = 0;
    pOut->_m [3][2] = 0; pOut->_m [3][3] = 1;
    return pOut;
}

#pragma argsused
D3DXMATRIX* D3DXMatrixLookAtLH (D3DXMATRIX* pOut, D3DXVECTOR3* pos, D3DXVECTOR3* target,
                                D3DXVECTOR3* y)
{
    D3DXVECTOR3 zaxis = *target - *pos;
    D3DXVec3Normalize (&zaxis, &zaxis);

    D3DXVECTOR3 xaxis;
```

```
    D3DXVec3Cross (&xaxis, y, &zaxis);
    D3DXVec3Normalize (&xaxis, &xaxis);

    D3DXVECTOR3 yaxis;
    D3DXVec3Cross (&yaxis, &zaxis, &xaxis);
//  D3DXVec3Normalize (&yaxis, &yaxis);

    pOut->_m [0][0] = xaxis.x; pOut->_m [0][1] = yaxis.x;
    pOut->_m [0][2] = zaxis.x; pOut->_m [0][3] = 0;

    pOut->_m [1][0] = xaxis.y; pOut->_m [1][1] = yaxis.y;
    pOut->_m [1][2] = zaxis.y; pOut->_m [1][3] = 0;

    pOut->_m [2][0] = xaxis.z; pOut->_m [2][1] = yaxis.z;
    pOut->_m [2][2] = zaxis.z; pOut->_m [2][3] = 0;

    pOut->_m [3][0] = -(xaxis * *pos); pOut->_m [3][1] = -(yaxis * *pos);
    pOut->_m [3][2] = -(zaxis * *pos); pOut->_m [3][3] = 1;
    return pOut;
}

#pragma argsused
D3DXMATRIX* D3DXMatrixPerspectiveFovLH (D3DXMATRIX* pOut, float angle, float aspect, float znear,
                                        float zfar)
{
    D3DXMatrixIdentity (pOut);

    float h = cos (angle / 2) / sin (angle / 2);
    float w = h / aspect;

    pOut->_m [0][0] = w / (znear);
    pOut->_m [1][1] = h / (znear);
    pOut->_m [2][2] = zfar / (zfar - znear);
    pOut->_m [3][2] = znear * zfar / (znear - zfar);
    pOut->_m [2][3] = 1;
    pOut->_m [3][3] = 0;
    return pOut;
}

D3DXMATRIX* D3DXMatrixTranspose (D3DXMATRIX* pOut, D3DXMATRIX* pMat)
{
    D3DXMATRIX tmp (
    pMat->_m [0][0], pMat->_m [1][0], pMat->_m [2][0], pMat->_m [3][0],
    pMat->_m [0][1], pMat->_m [1][1], pMat->_m [2][1], pMat->_m [3][1],
    pMat->_m [0][2], pMat->_m [1][2], pMat->_m [2][2], pMat->_m [3][2],
    pMat->_m [0][3], pMat->_m [1][3], pMat->_m [2][3], pMat->_m [3][3]);
    *pOut = tmp;
    return pOut;
}

float Det (D3DXMATRIX* pMat, int row, int col)
{
    float m[3][3];

    int my = 0;

    for (int y = 0; y < 4; y ++)
    {
        if (y != row)
        {
            int mx = 0;
            for (int x = 0; x < 4; x ++)
            {
                if (x != col)
                {
                    m [my][mx] = (*pMat)(y, x);
                    mx ++;
                }
            }
            my ++;
        }
    }

    float mul = 1.0f;
    if ((col + row) % 2 != 0)  mul = -1;
    return mul * (
    m [0][0] * m [1][1] * m [2][2] + m [1][0] * m [2][1] * m [0][2] +
    m [2][0] * m [0][1] * m [1][2] - m [0][0] * m [2][1] * m [1][2] -
    m [1][0] * m [0][1] * m [2][2] - m [2][0] * m [1][1] * m [0][2]);
```

```
}

#pragma argsused
D3DXMATRIX* D3DXMatrixInverse (D3DXMATRIX* pOut, float* pDet, D3DXMATRIX* pMat)
{
    D3DXMATRIX tmp;

    for (int y = 0; y < 4; y ++)
        for (int x = 0; x < 4; x ++)
            tmp (x, y) = Det (pMat, y, x);

    *pOut = tmp;

    return pOut;
}
```

Файл Vector.h

```
#ifndef VECTOR_H
#define VECTOR_H

#include "Direct3D\Matrix\Matrix.h"

class D3DXMATRIX;

class D3DXVECTOR3
{
public:
    D3DXVECTOR3 (float _x, float _y, float _z) : x (_x), y (_y), z (_z) { }
    D3DXVECTOR3 () { }
    float x, y, z;

    float operator* (const D3DXVECTOR3& vec) { return x * vec.x + y * vec.y + z * vec.z; }
    D3DXVECTOR3 operator- (const D3DXVECTOR3& vec) { return D3DXVECTOR3 (x - vec.x, y - vec.y, z - vec.z);}
    D3DXVECTOR3 operator- () { return D3DXVECTOR3 (-x, -y, -z); }
};

D3DXVECTOR3* D3DXVec3TransformCoord (D3DXVECTOR3* pOut, D3DXVECTOR3* pVec, D3DXMATRIX* pMat);
D3DXVECTOR3* D3DXVec3Normalize (D3DXVECTOR3* pOut, D3DXVECTOR3* pVec);
D3DXVECTOR3* D3DXVec3Cross (D3DXVECTOR3* pOut, D3DXVECTOR3* pVec1, D3DXVECTOR3* pVec2);

#endif
```

Файл Vector.cpp

```
#include "Direct3D\Vector\Vector.h"
#include <math.h>

D3DXVECTOR3* D3DXVec3TransformCoord (D3DXVECTOR3* pOut, D3DXVECTOR3* pVec, D3DXMATRIX* pMat)
{
    float x = pVec->x * pMat->_m [0][0] + pVec->y * pMat->_m [1][0] + pVec->z * pMat->_m [2][0] + pMat->_m [3][0];
    float y = pVec->x * pMat->_m [0][1] + pVec->y * pMat->_m [1][1] + pVec->z * pMat->_m [2][1] + pMat->_m [3][1];
    float z = pVec->x * pMat->_m [0][2] + pVec->y * pMat->_m [1][2] + pVec->z * pMat->_m [2][2] + pMat->_m [3][2];
    float w = pVec->x * pMat->_m [0][3] + pVec->y * pMat->_m [1][3] + pVec->z * pMat->_m [2][3] + pMat->_m [3][3];

    if (w != 0) {  x /= w; y /= w; z /= w; }

    pOut->x = x;    pOut->y = y;    pOut->z = z;
    return pOut;
}

D3DXVECTOR3* D3DXVec3Normalize (D3DXVECTOR3* pOut, D3DXVECTOR3* pVec)
{
    float length = sqrt (pVec->x * pVec->x + pVec->y * pVec->y +
                         pVec->z * pVec->z);
    if (length != 0)
    {
        pOut->x = pVec->x / length;
        pOut->y = pVec->y / length;
        pOut->z = pVec->z / length;
    }
    return pOut;
}
```

```
D3DXVECTOR3* D3DXVec3Cross (D3DXVECTOR3* pOut, D3DXVECTOR3* pVec1, D3DXVECTOR3* pVec2)
{
    float x = pVec1->y * pVec2->z - pVec1->z * pVec2->y;
    float y = pVec1->z * pVec2->x - pVec1->x * pVec2->z;
    float z = pVec1->x * pVec2->y - pVec1->y * pVec2->x;

    pOut->x = x;
    pOut->y = y;
    pOut->z = z;

    return pOut;
}
```

Файл WinLib.h

```
#ifndef WINLIB_H
#define WINLIB_H

#include "WinLib\WinLib2.h"
#include "Direct3D\IIB\IIB.h"
#include "Direct3D\IDevice\IDevice.h"
#include "Direct3D\ID3D\ID3D.h"
#include "Direct3D\IVB\IVB.h"
#include "Direct3D\Matrix\Matrix.h"
#include "Direct3D\Vector\Vector.h"

#endif
```

Файл WinLib2.h

```
#ifndef WIN_LIBRARY2_H
#define WIN_LIBRARY2_H

#include <memory.h>
#include <string.h>

#ifndef NULL
#define NULL 0
#endif

#define DECLARE_HANDLE(n) struct tag##n { char unused; };\
    typedef tag##n* n;

#define ZeroMemory(n, l) memset (n, l, 0)

DECLARE_HANDLE (HWND);
DECLARE_HANDLE (HINSTANCE);
DECLARE_HANDLE (HICON);
DECLARE_HANDLE (HCURSOR);
DECLARE_HANDLE (HBRUSH);
DECLARE_HANDLE (HMENU);
DECLARE_HANDLE (HDC);

typedef unsigned long LRESULT;
typedef unsigned long HRESULT;
typedef unsigned int UINT;
typedef int INT;
typedef long LONG;
typedef float FLOAT;
typedef unsigned long WPARAM;
typedef unsigned long LPARAM;

typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned long DWORD;

typedef const char* LPCSTR;
typedef char* LPSTR;

typedef unsigned char BOOL;
#define TRUE 1
```

```
#define FALSE 0

#define CALLBACK
#define APIENTRY

#define WM_DESTROY 0x0002
#define WM_QUIT    0x0012
#define WM_CHAR    0x0003
#define WM_KEYDOWN 0x0004
#define WM_PAINT   0x0008

struct POINT
{
    LONG x, y;
};

struct MSG
{
    HWND        hwnd;
    UINT        message;
    WPARAM      wParam;
    LPARAM      lParam;
    DWORD       time;
    POINT       pt;
};
typedef MSG* LPMSG;

typedef LRESULT CALLBACK (*WNDPROC)(HWND, UINT, WPARAM, LPARAM);

struct WNDCLASS
{
    UINT        style;
    WNDPROC     lpfnWndProc;
    int         cbClsExtra;
    int         cbWndExtra;
    HINSTANCE   hInstance;
    HICON       hIcon;
    HCURSOR     hCursor;
    HBRUSH      hbrBackground;
    LPCSTR      lpszMenuName;
    LPCSTR      lpszClassName;
};

#define PM_NOREMOVE 0x0000

typedef void* LPVOID;
typedef void VOID;

struct RECT { LONG left, top, right, bottom; };

INT APIENTRY WinMain (HINSTANCE, HINSTANCE, LPSTR, int);
HRESULT RegisterClass (WNDCLASS* lpWC);
HWND CreateWindowEx (UINT exStyle, LPCSTR lpClassName, LPCSTR lpWndName, UINT Style, int x, int y,
                     int width, int height, HWND hParent, HMENU hMenu, HINSTANCE hInstance,
                     LPVOID param);
void ShowWindow (HWND hwnd, UINT param);
void UpdateWindow (HWND hwnd);

BOOL GetMessage  (MSG* lpMsg, HWND hwnd, UINT minFilter, UINT maxFilter);
BOOL PeekMessage (MSG* lpMsg, HWND hwnd, UINT minFilter, UINT maxFilter, UINT bRemove);

void TranslateMessage (MSG* lpMsg);
void DispatchMessage (MSG* lpMsg);

HRESULT DefWindowProc (HWND hwnd, UINT nMsg, WPARAM wParam, LPARAM lParam);

VOID PostQuitMessage (int nExitCode);

HCURSOR LoadCursor (HINSTANCE, LPCSTR);
HICON LoadIcon   (HINSTANCE, LPCSTR);

void GetClientRect (HWND, RECT* lpRect);

#define IDC_ARROW  (LPSTR)0
#define IDI_APPLICATION (LPSTR)1

#define WS_EX_OVERLAPPEDWINDOW 2
#define WS_OVERLAPPEDWINDOW    0x0008
```

```
#define WS_POPUP           0x0001
#define WS_CAPTION         0x0002
#define WS_SYSMENU         0x0004

#define SW_SHOW               4
#define CS_DBLCLKS            5

#define FAILED(n) ((n) < 0)

#define VK_LEFT  75
#define VK_RIGHT 77

struct PAINTSTRUCT { int a; };

HDC BeginPaint (HWND hwnd, PAINTSTRUCT* pps);
void EndPaint (HWND hwnd, PAINTSTRUCT* pps);

void DestroyWindow (HWND hWnd);

#endif
```

Файл WinLib.cpp

```
//#define BGI

#include "WinLib\Winlib2.h"
#include <conio.h>
#include "graph\graph.h"
#include "dac\dac.h"

#include <stdlib.h>
#ifdef BC
#include <graphics.h>
#endif

WNDCLASS *CurWC = NULL;

BOOL bExit = FALSE;

#pragma argsused
HRESULT RegisterClass (WNDCLASS* lpWC) { CurWC = lpWC; return 0; }

#pragma argsused
HWND CreateWindowEx (UINT exStyle, LPCSTR lpClassName, LPCSTR lpWndName, UINT Style, int x, int y,
                     int width, int height, HWND hParent, HMENU hMenu, HINSTANCE hInstance,
                     LPVOID param) {  return (HWND)100; }
#pragma argsused
void ShowWindow (HWND hwnd, UINT param) { }

#pragma argsused
void UpdateWindow (HWND hwnd) { }

#pragma argsused
BOOL GetMessage  (MSG* lpMsg, HWND hwnd, UINT minFilter, UINT maxFilter) { return TRUE; }

#pragma argsused
BOOL PeekMessage (MSG* lpMsg, HWND hwnd, UINT minFilter, UINT maxFilter, UINT bRemove)
{
    lpMsg->message = 0;
    if (bExit)
    {
        lpMsg->message = WM_QUIT;
        return TRUE;
    }
    if (kbhit ())
    {
        int key = getch ();
        switch (key) {
        case 27:
            lpMsg->message = WM_QUIT;
            return TRUE;
        default:
            lpMsg->message = WM_KEYDOWN;
            lpMsg->wParam = key;
            CurWC->lpfnWndProc (hwnd, lpMsg->message, lpMsg->wParam, lpMsg->lParam);
            return TRUE;
```

```
        };
    }
    return FALSE;
}

#pragma argsused
void TranslateMessage (MSG* lpMsg) { }
#pragma argsused
void DispatchMessage  (MSG* lpMsg) { }

#pragma argsused
HRESULT DefWindowProc (HWND hwnd, UINT nMsg, WPARAM wParam, LPARAM lParam) { return 0; }

#pragma argsused
VOID PostQuitMessage (int nExitCode) { }

#pragma argsused
HCURSOR LoadCursor (HINSTANCE, LPCSTR) { return (HCURSOR)0; }
#pragma argsused
HICON LoadIcon   (HINSTANCE, LPCSTR) { return (HICON)0; }

int main ()
{
#ifndef BGI
    StartGraph ();
#else
    int gD = installuserdriver ("SVGA256", NULL), gM = 4;
    initgraph (&gD, &gM, "P:\\prog\\bgi");
#endif

    WinMain ((HINSTANCE)100, (HINSTANCE)0, NULL, 0);
    return 0;
}

void GetClientRect (HWND, RECT* lpRect)
{
    lpRect->left = lpRect->top = 0;
    lpRect->right = 1024;
    lpRect->bottom = 768;
}

HDC BeginPaint (HWND hwnd, PAINTSTRUCT* pps) { return NULL; }
void EndPaint (HWND hwnd, PAINTSTRUCT* pps) { }

void DestroyWindow (HWND hWnd)
{
    bExit = TRUE;
}
```

П.4. Borland-версия библиотеки

Файл BackBuf.h

```
#ifndef BACKBUF_H
#define BACKBUF_H

#include "WinLib\WinLib2.h"
#include "xmm\xmm.h"

#define NULL 0

class CBackBuffer
{
protected:
    WORD m_hHandler;

    BYTE* m_TmpData;
public:
    CBackBuffer ();
    virtual void Alloc ();
    virtual void GetScanLine (BYTE** lpBuffer, int line);
    virtual void SetScanLine (BYTE** lpBuffer, int line);
    virtual void Dealloc ();
```

```
    virtual void Clear (int color, RECT* pRect = NULL);

    virtual void Flip (RECT* pRect = NULL);

    operator WORD () { return m_hHandler; }

    int  GetPixel (int x, int y);
    void SetPixel (int x, int y, int color);

    ~CBackBuffer ();
};

#endif
```

Файл BackBuf.cpp

```
#include "BackBuf\BackBuf.h"
#include "graph\graph.h"
#include <assert.h>
#include <stdio.h>

#define EVEN(n) (int)((n) - ((n) % 2))

CBackBuffer::CBackBuffer ()
{
    XMM_Initialize ();
    m_TmpData = new BYTE [1024];
}

void CBackBuffer::Alloc ()
{
    m_hHandler = XMM_AllocateBlock ((int)((1024L * 768L) >> 10));
}

void CBackBuffer::Clear (int color, RECT* pRect)
{
    byte lpTmp [1024];
    if (!pRect)
    {
        memset (lpTmp, (byte)color, 1024);

        for (int c = 0; c < 768; c ++)
            XMM_memcpy (m_hHandler, ((long)c) << 10, lpTmp, 1024);
    }
    else
    {
        memset (lpTmp, (byte)color, (int)(pRect->right - pRect->left));
        int length = EVEN(pRect->right - pRect->left) ==
            (int)(pRect->right - pRect->left) ?
            (int)(pRect->right - pRect->left) : EVEN (pRect->right - pRect->left);

        for (LONG c = pRect->top; c <= pRect->bottom; c ++)
            XMM_memcpy (m_hHandler, c * 1024L + (LONG)EVEN (pRect->left), (byte*)lpTmp, length);
    }
}

void CBackBuffer::Dealloc ()
{
    XMM_FreeBlock (m_hHandler);
}

void CBackBuffer::GetScanLine (byte** lpBuffer, int line)
{
    XMM_memcpy (m_TmpData, m_hHandler, ((long)line) << 10, 1024L);
    *lpBuffer = m_TmpData;
}

void CBackBuffer::SetScanLine (byte** lpBuffer, int line)
{
    XMM_memcpy (m_hHandler, ((long)line) << 10, *lpBuffer, 1024L);
}

void CBackBuffer::Flip (RECT* pRect)
{
/*  if (!pRect)
```

```
    {
        for (int c = 0; c < 12; c ++)
        {
            SetPage (c);
            XMM_memcpy ((byte far*)0xa0000000L, m_hHandler, ((long)c) << 16, 65536L);
        }
    }
    else
*/  {
/*      FILE* pDump = fopen ("dump.txt", "w");
        fclose (pDump);
*/
        for (DWORD y = 0; y < 1; y ++)
        {
            SetPage (y / 64);

            byte far* pAddr = (byte far*)0xa0000000L;
            XMM_memcpy (pAddr,
                m_hHandler, y * 1024L, 1024);
        }
    }

}

int  CBackBuffer::GetPixel (int x, int y)
{
    short Buffer;
    XMM_memcpy ((byte far*)&Buffer, m_hHandler, y * 1024 + EVEN(x), 2);
    if (x % 2 == 1) return Buffer << 8;
    else return (int)(char)Buffer;
}

void CBackBuffer::SetPixel (int x, int y, int color)
{
    short Buffer;
    XMM_memcpy ((byte far*)&Buffer, m_hHandler, y * 1024L + EVEN(x), 2);
    Buffer &= 0xff << (!(x % 2)) * 8;
    Buffer |= (char)color << (x % 2) * 8;
    XMM_memcpy (m_hHandler, y * 1024L + EVEN(x), (byte far*)&Buffer, 2);
}

CBackBuffer::~CBackBuffer ()
{
    delete[] m_TmpData;
}
```

Файл IDev.cpp

```
#include "Direct3D\IDevice\IDevice.h"

void IDirect3DDevice8::SetPixel (Line line, BYTE* colorplace, WORD* zplace, int x, int y)
{
    if (zplace [x] > line.z * 65535U)
    {
        colorplace [x] =
            GetIndex (line.color.r, line.color.g, line.color.b);
        zplace [x] = (WORD)(line.z * 65535U);
    }
}
```

Файл ZBuf.h

```
#ifndef ZBUFFER_H
#define ZBUFFER_H

#include "WinLib2.h"

class CZBuffer
{
protected:
    unsigned int m_hHandler;

    WORD* m_TmpData;
```

```
public:
    CZBuffer ();
    virtual void Alloc ();
    virtual void GetScanLine (WORD** lpBuffer, int line);
    virtual void SetScanLine (WORD** lpBuffer, int line);
    virtual void Dealloc ();

    virtual void Clear (RECT* pRect = NULL);

    operator unsigned int () { return m_hHandler; }

    unsigned short GetPixel (int x, int y);
    void SetPixel (int x, int y, unsigned int z);

    ~CZBuffer ();
};

#endif
```

Файл Zbuf.cpp

```
#include "ZBuf\ZBuf.h"
#include "xmm\xmm.h"

CZBuffer::CZBuffer ()
{
    XMM_Initialize ();
    m_TmpData = new WORD [1024];
}

void CZBuffer::Alloc ()
{
    m_hHandler = XMM_AllocateBlock ((int)((1024L * 768L * 2L) >> 10));
}

void CZBuffer::GetScanLine (WORD** lpBuffer, int line)
{
    XMM_memcpy ((byte*)m_TmpData, m_hHandler, ((long)line) << 11, 2048L);
    *lpBuffer = m_TmpData;
}

void CZBuffer::SetScanLine (WORD** lpBuffer, int line)
{
    XMM_memcpy (m_hHandler, ((long)line) << 11, (byte*)(*lpBuffer), 2048L);
}

void CZBuffer::Dealloc ()
{
    XMM_FreeBlock (m_hHandler);
}

void CZBuffer::Clear (RECT* pRect)
{
    byte lpTmp [2048];
    memset (lpTmp, (char)0xff, 2048);

    if (!pRect)
        for (LONG c = 0; c < 768; c ++)
            XMM_memcpy (m_hHandler, c * 2048L, lpTmp, 2048);
    else
        for (LONG c = pRect->top; c <= pRect->bottom; c ++)
            XMM_memcpy (m_hHandler, c * 2048L + 2 * pRect->left, lpTmp, 2 * (pRect->right - pRect-
>left));
}

unsigned short CZBuffer::GetPixel (int x, int y)
{
    unsigned short Buffer;
    XMM_memcpy ((byte far*)&Buffer, m_hHandler, y * 2048L + x * 2, 2);
    return Buffer;
}

void CZBuffer::SetPixel (int x, int y, unsigned int z)
{
    XMM_memcpy (m_hHandler, y * 2048L + x * 2, (byte far*)&z, 2);
}
```

```
CZBuffer::~CZBuffer ()
{
    delete[] m_TmpData;
}
```

Файл Ctrls.cpp

```
#include "Ctrls.h"

void CreateControls ()
{
}

void ProcessControls (char ch)
{
}

void DeleteControls ()
{
}

void OnPaint (HDC hdc)
{
}
```

П.5. GCC-версия библиотеки

Файл BackBuf.h

```
#ifndef BACKBUF_H
#define BACKBUF_H

#include "WinLib\WinLib2.h"
#include <grx20.h>

#define NULL 0

class CBackBuffer
{
protected:
    GrContext* m_pContext;
public:
    CBackBuffer ();
    virtual void Alloc ();
    virtual void GetScanLine (BYTE** lpBuffer, int line);
    virtual void SetScanLine (BYTE** lpBuffer, int line);
    virtual void Dealloc ();

    virtual void Clear (int color, RECT* pRect = NULL);

    virtual void Flip (RECT* pRect = NULL);

    int  GetPixel (int x, int y);
    void SetPixel (int x, int y, int color);

    GrContext* Context () { return m_pContext; }

    ~CBackBuffer ();
};

#endif
```

Файл BackBuf.cpp

```
#include "BackBuf\BackBuf.h"
#include <grx20.h>

CBackBuffer::CBackBuffer () : m_pContext (NULL)
{
}

void CBackBuffer::Alloc ()
{
    m_pContext = GrCreateContext (1024, 768, NULL, NULL);
    GrSetContext (m_pContext);
}

void CBackBuffer::GetScanLine (BYTE** lpBuffer, int line)
{
    const GrContext* pCur = GrCurrentContext ();
    *lpBuffer = NULL;
}

void CBackBuffer::SetScanLine (BYTE** lpBuffer, int line)
{
}

void CBackBuffer::Dealloc ()
{
    GrSetContext (NULL);
    GrDestroyContext (m_pContext);
    m_pContext = NULL;
}

void CBackBuffer::Clear (int color, RECT* pRect)
{
    GrFilledBox (10, 10, 1014, 618, color);
}

void CBackBuffer::Flip (RECT* pRect)
{
    GrSetContext (NULL);
    if (!pRect)
        GrBitBlt (NULL, 0, 0, m_pContext, 0, 0, 1024, 768, GrWRITE);
    else
        GrBitBlt (NULL, pRect->left, pRect->top,
        m_pContext, pRect->left, pRect->top, pRect->right, pRect->bottom,
        GrWRITE);
    GrSetContext (m_pContext);
}

int  CBackBuffer::GetPixel (int x, int y)
{
    return GrPixel (x, y);
}

void CBackBuffer::SetPixel (int x, int y, int color)
{
    if (x > 10 && y > 10 && x < 1014 && y < 618)
        GrPlot (x, y, color);
}

CBackBuffer::~CBackBuffer ()
{
}
```

Файл DAC.h

```
#ifndef DAC_H
#define DAC_H

void SetDacReg (int reg, char r, char g, char b);

#endif
```

Файл DAC.cpp

```
#include "dac\dac.h"
#include <grx20.h>

void SetDacReg (int reg, char r, char g, char b)
{
    GrSetColor (reg, r * 4, g * 4, b * 4);
}
```

Файл Graph.h

```
#ifndef GRAPH_H
#define GRAPH_H

void StartGraph ();

#endif
```

Файл Graph.cpp

```
#include "graph\graph.h"
#include <grx20.h>

void StartGraph ()
{
    GrSetMode (GR_width_height_color_graphics, (int)1024, (int)768, (GrColor)256);
}
```

Файл IDev.cpp

```
#include <Direct3D\IDevice\IDevice.h>

void IDirect3DDevice8::SetPixel (Line line, BYTE* colorplace, WORD* zplace, int x, int y)
{
    if (zplace [x] > line.z * 65535U)
    {
        m_BackBuffer.SetPixel (x, y, GetIndex
                               (line.color.r, line.color.g, line.color.b));
/*        ((DWORD*)colorplace) [offset] = (DWORD)GetIndex
                               (line.color.r, line.color.g, line.color.b);
*/      zplace [x] = (WORD)(line.z * 65535U);
    }
}
```

Файл ZBuf.h

```
#ifndef ZBUFFER_H
#define ZBUFFER_H

#include "WinLib2.h"

class CZBuffer
{
protected:
    WORD* m_lpData;
public:
    CZBuffer ();
    virtual void Alloc ();
    virtual void GetScanLine (WORD** lpBuffer, int line);
    virtual void SetScanLine (WORD** lpBuffer, int line);
    virtual void Dealloc ();

    virtual void Clear (RECT* pRect = NULL);
```

```
    WORD GetPixel (int x, int y);
    void SetPixel (int x, int y, WORD z);

    ~CZBuffer ();
};

#endif
```

Файл ZBuf.cpp

```
#include "ZBuf\ZBuf.h"

CZBuffer::CZBuffer () : m_lpData (NULL)
{
}

void CZBuffer::Alloc ()
{
    m_lpData = new WORD [1024L * 768L];
}

void CZBuffer::GetScanLine (WORD** lpBuffer, int line)
{
    *lpBuffer = &m_lpData [line * 1024L];
}

void CZBuffer::SetScanLine (WORD** lpBuffer, int line)
{
}

void CZBuffer::Dealloc ()
{
    delete[] m_lpData;
    m_lpData = NULL;
}

void CZBuffer::Clear (RECT* pRect)
{
    if (!pRect)
        memset (m_lpData, 0xff, 1024L * 768L * 2L);
    else
        for (UINT y = pRect->top; y <= pRect->bottom; y ++)
            memset (&m_lpData [y * 1024 + pRect->left], 0xff, (pRect->right - pRect->left) * 2);
}

WORD CZBuffer::GetPixel (int x, int y)
{
    if (x > 10 && y > 10 && x < 1014 && y < 618)
        return m_lpData [y * 1024L + x];
    return 0;
}

void CZBuffer::SetPixel (int x, int y, WORD z)
{
    if (x > 10 && y > 10 && x < 1014 && y < 618)
        m_lpData [y * 1024L + x] = z;
}

CZBuffer::~CZBuffer ()
{
}
```

Файл Ctrls.cpp

```
#include "ctrls.h"
#include <grx20.h>
#include <direct3d\idevice\idevice.h>
#include <stdio.h>
#define NULL 0
GrColor* pColors = NULL;
```

```
class IDirect3DDevice8;

extern IDirect3DDevice8* lpDevice;

extern HWND hWnd;
extern int CurrentObject;
extern BOOL bChanged;
extern int FillMode;

void DrawButton (int x1, int y1, int w, int h, char* capt, BOOL bPushed)
{
    int x2 = x1 + w;
    int y2 = y1 + h;

    int color1 = 15;
    int color2 = 8;
    if (bPushed)
    {
        color1 = 8;
        color2 = 15;
    }

    GrVLine (x1, y1, y2, color1);
    GrVLine (x1 + 1, y1 + 1, y2 - 1, color1);
    GrVLine (x2, y1, y2, color2);
    GrVLine (x2 - 1, y1 + 1, y2 - 1, color2);

    GrHLine (x1, x2, y1, color1);
    GrHLine (x1 + 1, x2 - 1, y1 + 1, color1);
    GrHLine (x1, x2, y2, color2);
    GrHLine (x1 + 1, x2 - 1, y2 - 1, color2);

    GrFilledBox (x1 + 2, y1 + 2, x2 - 2, y2 - 2, 7);

    int StrWidth = strlen (capt) * 8;
    int StrHeight = 16;
    GrTextXY(x1 + (x2 - x1 - StrWidth) / 2, y1 + (y2 - y1 - StrHeight) / 2, capt, 0, 7);

}

char* capt [] = {
      "1 -- Cube",
      "2 -- Cylinder",
      "3 -- Sphere",
      "4 -- Lamp",
      "5 -- Exit",
      "6 -- Solid",
      "7 -- Wireframe"
};

void CreateControls ()
{
    GrSetContext (lpDevice->BackBuffer ()->Context ());
    GrVLine (0, 0, 767, 15);
    GrVLine (1, 1, 766, 15);
    GrHLine (0, 1023, 0, 15);
    GrHLine (1, 1022, 1, 15);
    GrVLine (1023, 0, 767, 8);
    GrVLine (1022, 1, 766, 8);
    GrHLine (0, 1023, 767, 8);
    GrHLine (1, 1022, 766, 8);

    GrVLine (8, 8, 620, 8);
    GrVLine (9, 9, 619, 8);
    GrHLine (8, 1016, 8, 8);
    GrHLine (9, 1015, 9, 8);
    GrVLine (1016, 8, 620, 15);
    GrVLine (1015, 9, 619, 15);
    GrHLine (8, 1016, 620, 15);
    GrHLine (9, 1015, 619, 15);

    GrFilledBox (2, 2, 7, 765, 7);
    GrFilledBox (1017, 2, 1021, 765, 7);
    GrFilledBox (2, 2, 1021, 7, 7);
    GrFilledBox (8, 621, 1016, 765, 7);

    DrawButton (30,  618 + 40, 120, 40, "1 -- Cube",    FALSE);
    DrawButton (160, 618 + 40, 120, 40, "2 -- Cylinder",FALSE);
    DrawButton (290, 618 + 40, 120, 40, "3 -- Sphere",  FALSE);
```

```
    DrawButton (420, 618 + 40, 120, 40, "4 -- Lamp",    FALSE);
    DrawButton (30,  618 + 100, 120, 40, "5 -- Exit",    FALSE);
    DrawButton (160, 618 + 100, 120, 40, "6 -- Solid",   FALSE);
    DrawButton (290, 618 + 100, 120, 40, "7 -- Wireframe",FALSE);

}

int OldButton = -1;

void ProcessControls (char ch)
{
    int num = ch - '1';

    if (num < 0 || num > 6) return;

    if (num != OldButton && num < 4)
        DrawButton (num * 130 + 30, 618 + 40, 120, 40, capt [num], TRUE);
    else
        DrawButton ((num - 4) * 130 + 30, 618 + 100, 120, 40, capt [num], TRUE);

    if (OldButton != -1 && num != OldButton && OldButton < 4)
       DrawButton (OldButton * 130 + 30, 618 + 40, 120, 40, capt [OldButton], FALSE);
    else
       DrawButton ((OldButton - 4) * 130 + 30, 618 + 100, 120, 40, capt [OldButton], FALSE);

    if (num >= 0 && num < 4)
    {
        CurrentObject = num + 1;
        bChanged = TRUE;
    }
    if (num == 5)       FillMode = D3DFILL_SOLID;
    if (num == 6)       FillMode = D3DFILL_WIREFRAME;

    OldButton = num;

    if (num == 4) DestroyWindow (hWnd);
}

void DeleteControls ()
{

}

void OnPaint (HDC hdc)
{
}
```

Здравствуйте, уважаемая комиссия. Меня зовут **Попов Владимир**.

СЛАЙД 1

Разрешите представить мой проект: **МУЛЬТИПЛАТФОРМЕННАЯ РЕАЛИЗАЦИЯ КОМПЬЮТЕРНОЙ ГРАФИКИ**.

ЦЕЛЬЮ проекта было создание набора библиотек, необходимых для компиляции под **РАЗЛИЧНЫЕ ОПЕРАЦИОННЫЕ СИСТЕМЫ**: с поддержкой различных компиляторов.

Версия для **DOS** предполагает создание собственного **КОНВЕЙЕРА РЕНДЕРИНГА.**

Для создания приложения, **РАБОТАЮЩЕГО** под **DOS**, необходимо было **ЭМУЛИРОВАТЬ** часть функций Windows API.

Также было необходимо **ЭМУЛИРОВАТЬ ЧАСТЬ ИНТЕРФЕЙСА MICROSOFT DIRECT3D**.

Не менее важная задача — создать **ДЕМОНСТРАЦИОННУЮ ПРОГРАММУ**.

СЛАЙД 2

Рассмотрим **АРХИТЕКТУРУ** программы. Программа-приложение **ОПИРАЕТСЯ НА РЕСУРСЫ БИБЛИОТЕКИ**. Если приложение компилируется под **WINDOWS,** то используется **DIRECT3D.** Если программа компилируется под **DOS**, то используется **ЭМУЛЯЦИЯ** Direct3D — **КОНВЕЙЕРА РЕНДЕРИНГА** и его **API.** Конвейер опирается на **ГРАФИЧЕСКИЕ БИБЛИОТЕКИ**, которые **ЗАВИСЯТ ОТ КОМПИЛЯТОРА**.

Здесь представлена **СТРУКТУРА ПРОЕКТА. ФАЙЛЫ РАССОРТИРОВАНЫ** по зависимости от платформы и компилятора. Низкоуровневая графика в **GCC** использует библиотеку **GRX**, а в **BORLAND** пришлось написать **СОБСТВЕННУЮ,** поддерживающую режимы **SVGA.**

СЛАЙД 3

ЯДРОМ ПРОГРАММЫ ЯВЛЯЕТСЯ КОНВЕЙЕР рендеринга. Он разделен на несколько этапов. Для сокрытия пересекающихся поверхностей использу-

ется алгоритм **Z-БУФЕРА,** основанный на анализе **Z-КООРДИНАТ** рисуемых пикселей.

Для **СМЕНЫ КАДРОВ** изображения используется **ДВОЙНАЯ БУФЕРИЗАЦИЯ**. Программа стирает и рисует новое изображение в **БЛОКЕ** оперативной **ПАМЯТИ,** а затем **КОПИРУЕТ** результат на экран.

Для эмуляции интерфейса Direct3D были создано **НЕСКОЛЬКО КЛАССОВ**, аналогичных классам Direct3D.

Они предназначены для безопасного создания и удаления объектов, обеспечения рендеринга, хранения вершин и их нормалей.

СЛАЙД 4

Программа может показывать различные **ФИГУРЫ** и их **ПРОВОЛОЧНЫЕ МОДЕЛИ**. Она позволяет выводить **ИЗОБРАЖЕНИЯ ПРИМИТИВОВ** (куба, цилиндра, сферы) а также простой **МОДЕЛИ ПРОЖЕКТОРА**.

Для достижения реалистичного освещения создается **ПАЛИТРА ЦВЕТОВ**. Используется режим с 256 цветами.

Для вычисления освещения треугольника используется модель Ламберта.

СЛАЙД 5

ИТАК, был создан набор **ПЕРЕНОСИМЫХ БИБЛИОТЕК,** позволяющий использовать элементы технологии DirectX.

Был создан **КОНВЕЙЕР РЕНДЕРИНГА**.

Было **ЭМУЛИРОВАНО** некоторое подмножество **ФУНКЦИЙ WINDOWS API И MICROSOFT DIRECT3D.**

Были созданы версии библиотеки под 2 платформы и 3 компилятора.

Была создана тестовая программа.

Теперь разрешите **ПЕРЕЙТИ К ДЕМОНСТРАЦИИ.**

Вначале рассмотрим DOS версию программы, скомпилированную под GCC.

После запуска программы открывается окно, на котором мы видим **ИЗОБРАЖЕНИЕ** вращающейся **МОДЕЛИ КУБА**.

С помощью **КНОПОК** мы можем **ПЕРЕКЛЮЧАТЬ** изображения объектов. Например, нажмем кнопку **2**. Изображение сменилось на **ИЗОБРАЖЕНИЕ ЦИЛИНДРА**. Нажмем **КЛАВИШУ 6** — изображение сменяется на изображение его **ПРОВОЛОЧНОЙ МОДЕЛИ**. Можно посмотреть и **ДРУГИЕ ОБЪЕКТЫ**.

Теперь запустим **WINDOWS-ВЕРСИЮ**, использующую возможности аппаратного ускорения DirectX. Конечно, скорость работы программы увеличивается, однако, при **ОТСУТСТВИИ АППАРАТНОГО УСКОРЕНИЯ** и на одной и той же машине мы получали даже **БОЛЬШУЮ СКОРОСТЬ** работы у **DOS32**-приложения.

ВЫЙТИ из программы можно при нажатии на клавишу **5** или клавишу **ESC**.

СПАСИБО ЗА ВНИМАНИЕ. Если есть **ВОПРОСЫ**, я готов на них ответить.