**Содержание**

Введение

1. Программирование с использованием библиотеки OреnG1

1.1 Принципы OреnG1

1.2 Основные приоритеты OреnG1

1.3 Архитектура OреnG1

1.4 Описание математического аппарата и проекции

1.5 Материалы

2. Разработка приложения для построения динамической изображения трехмерной модели объекта «Паровоз»

2.1 Разработка процедуры визуализации трехмерной сцены

2.2 Разработка интерфейса пользователя

2.3 Разработка подсистемы управлениями событиями

3. Информационное и программное обеспечение

3.1 Общие сведения о программе

3.2 Функциональное назначение

3.3 Логическая структура и функциональная декомпозиция проекта

3.4 Требования к техническому программному обеспечению

3.5 Руководство пользователя

4. Заключение и выводы

Список литературы

Приложение 1

Приложение 2

Приложение 3

Приложение 4

Приложение 5

**Введение**

Курсовой проект был создан с помощью среды разработки Visual Studio 6.0 на языке C++ c использованием библиотеки OреnG1.

Среда разработки Microsoft Visual Studio 6.0 — это набор инструментов и средств, предназначенных для помощи разработчикам программ любого уровня квалификации в решении сложных задач и создания новаторских решений. Роль Visual Studio заключается в том, чтобы улучшить процесс разработки и упростить разработку высокоэффективных программ.

В Visual Studio содержатся инструменты для всех этапов разработки программного обеспечения (разработка, тестирование, развертывание, интеграция и управления) и для разработчиков любого уровня квалификации, от новичков до опытных специалистов.

На базовом уровне, OреnG1 — это просто спецификация, то есть документ, описывающий набор функций и их точное поведение. Производители оборудования на основе этой спецификации создают реализации — библиотеки функций, соответствующих набору функций спецификации. Реализация использует возможности оборудования там, где это возможно. Если аппаратура не позволяет реализовать какую-либо возможность, она должна быть эмулирована программно. Производители должны пройти специфические тесты (conformance tests — тесты на соответствие) прежде чем реализация будет классифицирована как OреnG1 реализация. Таким образом, разработчикам программного обеспечения достаточно научиться использовать функции, описанные в спецификации, оставив эффективную реализацию последних разработчикам аппаратного обеспечения.

Эффективные реализации OреnG1 существуют для Windows, Unix-платформ, PlayStation 3 и Mac OS. Эти реализации обычно предоставляются изготовителями видеоадаптеров и активно используют возможности последних. Существуют также чисто программные реализации спецификации OреnG1, одной из которых является библиотека Mesa. Из лицензионных соображений Mesa является «неофициальной» реализацией OреnG1, хотя полностью с ней совместима на уровне кода.

OреnG1 ориентируется на следующие две задачи:

Скрыть сложности адаптации различных 3D-ускорителей, предоставляя разработчику единый API.

Скрыть различия в возможностях аппаратных платформ, требуя реализации недостающей функциональности с помощью программной эмуляции.

Основным принципом работы OреnG1 является получение наборов векторных графических примитивов в виде точек, линий и многоугольников с последующей математической обработкой полученных данных и построением растровой картинки на экране и/или в памяти. Векторные трансформации и растеризация выполняются графическим конвейером (graphics pipeline), который по сути представляет собой дискретный автомат. Абсолютное большинство команд OреnG1 попадают в одну из двух групп: либо они добавляют графические примитивы на вход в конвейер, либо конфигурируют конвейер на различное исполнение трансформаций.

# **1. Программирование с использованием библиотеки OреnG1**

## **1.1 Принципы OреnG1**

Основная идея OреnG1: графическая библиотека должна быть аппаратно независимой, но в то же время использовать аппаратные ускорители, если они доступны. Кроме того, этот язык с самого начала предусматривал механизм расширения и гибкости - по мере того как расширения становились "общепринятыми", они становились частью следующего релиза.

Для любого протокола, будь то сетевой протокол или язык описания сцен, важным является вопрос уровня абстракции - то есть того, на каком уровне работает данная система или протокол, что является входными данными и что выходными, какие компоненты будут взаимодействовать в качестве поставщиков и приемников данных. Говоря попросту, нужно определиться по вопросу "что мы делаем и чего не делаем".

Создатели OреnG1 планировали свой язык с явным намерением создать "виртуальный графический акселератор", так чтобы примитивы OреnG1 максимально соответствовали примитивам современных графических карт и требовали минимум кода для трансляции из одной системы команд в другую. Фактически большинство современных графических процессоров (обычно называемых видеокартами, хотя к видео они имеют лишь касательное отношение) напрямую воспринимают OреnG1 как язык входного уровня без какой-либо (или с минимумом) трансляции.

OреnG1 оперирует графическими примитивами "начального уровня", такими как точки трехмерного пространства (вертексы, вершины), отрезки прямых, выпуклые полигоны и растровые изображения. Поддерживаются аффинные и проективные преобразования, вычисление освещения. К "продвинутым" функциям можно отнести мэппинг текстур (натягивание битовых карт на трехмерные поверхности) и антиалиасинг (сглаживание цветовых переходов - как локальное, в рамках отдельного объекта, так и глобальное, по всей сцене). Предполагается, что приложение более высокого уровня будет выполнять операции, которых недостает в OреnG1,- например, декомпозицию невыпуклых полигонов.

С точки зрения программиста, OреnG1 - это система вызовов процедур с передачей им параметров, то есть этот язык представляет собой Call Level API. Ключевым моментом с точки зрения производительности, особенно в сетевом окружении, является наличие двух режимов: пошагового и пакетного. Пакетный группирует команды описания объектов и режимы в пакет, называемый списком отображения (display list). Подобная техника применяется, например, при описании страниц в PostScript или, в несколько другом контексте, при создании хранимых процедур в SQL. Смысл ясен - инкапсуляция функциональности (в данном случае - объектов для последующего многократного использования). При этом можно провести несколько оптимизаций: предварительно один раз проверить синтаксис, а также кэшировать готовый объект при передаче по сети на целевой машине.

Недостаток списка отображения проявляется при частом внесении в него изменений - при этом, понятно, ни один метод оптимизации не будет работать. Для работы с такими "изменчивыми" объектами и сценами в OреnG1 предусмотрен режим прямого отображения, когда каждое предложение интерпретируется в порядке поступления, не дожидаясь закрывающего список тега или какой-то подобной команды на отрисовку. В практике оба метода нашли широкое применение.

## **1.2 Основные приоритеты OреnG1**

При реализации авторы ставили себе пять ориентиров, важных с точки зрения получаемых результатов.

1.Производительность. С самого начала в OреnG1 была заложена "крайне желательная" возможность отрисовки динамических сцен. Для получения нужных результатов в систему введено множество параметров, или, как говорят, режимов рисования. Если некоторый режим или комбинация режимов на данном оборудовании не в состоянии обеспечить интерактивного взаимодействия и необходимой частоты обновления сцены, то пользователь или сама программа должны быть в состоянии отключать так много дополнительных функций, сколько нужно для получения "живой" картинки.

2.Ортогональность. По возможности все функции OреnG1 являются ортогональными, то есть независимыми. Вы можете использовать их в произвольной комбинации, например использование мэппинга не ограничивает возможностей применения светотени.

3.Полнота. Насколько это представляется возможным, OреnG1 соответствует набору функций, предоставляемому современными аппаратными средствами графической акселерации. OреnG1 старается избегать всего, что должно быть реализовано программно. С другой стороны, по крайней мере, гарантируется получение рабочей картинки, даже если производительность и не позволяет получить ее со всеми подробностями. То есть, если что-то работает на одной платформе, то этот же код будет работать и на другой - хотя, возможно, и с другим результатом.

4.Интероперабельность. В сетевом окружении важно передавать данные между разными платформами. Поэтому OреnG1 заранее ориентирован на работу в режиме клиент-сервер, даже если и клиент и сервер расположены на одном компьютере.

5.Расширяемость. Поскольку OреnG1 рассчитан на максимальное соответствие возможностям аппаратуры (а аппаратура, как известно, имеет тенденцию развиваться), то в OреnG1 также встроены механизмы включения новых функций. С другой стороны, нестабильный интерфейс затрудняет жизнь разработчиков, поэтому новые возможности накапливаются достаточное время и применяются согласованно с выходом новой версии.

## **1.3 Архитектура OреnG1**

Более глубоко погружаясь в технологию OреnG1, мы обнаружим стройную и гармоничную архитектуру, общий вид которой показан на рисунке.



Рисунок 1 – Архитектура OреnG1

Точно так же, как процессор имеет два типа конвейеров - для целочисленных вычислений и чисел с плавающей точкой, OреnG1 имеет два конвейера для пиксельных данных и вертексных операций, то есть для операций с векторными данными. Каждый из потоков обрабатывается отдельно, до тех пор пока это возможно - то есть до стадии мэппинга, когда пиксельные растры как фактуры "натягиваются" на плоскости и более сложные поверхности.

Первый этап - аппроксимация кривых и поверхностей вычислением полиномов от входных значений. Второй проход оперирует с примитивами типа точек, отрезков и полигонов - они преобразуются по правилам аффинных преобразований, совмещаются и сцена отсекается в подготовке к растрированию.

Растрирование в качестве результата создает список объектов (точек, отрезков и треугольников) в двумерной плоскости. Над отдельными объектами может быть выполнена операция раскрашивания, градиентной заливки или применения мэппинга, то есть наложения фактуры.

Готовые фрагменты окончательно обрабатываются перед тем, как они реально будут внесены в frame buffer. В частности, фрагменты сортируются зависимо от значений "глубины" - и эти значения сравниваются с известными на предмет рекомпозиции. Применение блендинга приводит к тому, что прозрачные фрагменты принимают цвет, состоящий из их собственного и цветов "ниже лежащих" фрагментов. Дополнительно может быть реализовано маскирование и другие эффекты.

Пиксельный процессор в ходе растрирования встраивает двумерные битовые фрагменты прямо в кадр. Часть готового кадра также может быть прочитана для повторного использования как массив пикселей - так что данные, отображаемые в буфере, могут стать частью других сцен.

## **1.4 Описание математического аппарата и проекции**

Движение графических объектов в пространстве осуществляется посредством аффинных преобразований. Любое аффинное преобразование в трехмерном пространстве может быть представлено в виде суперпозиции вращений, растяжений, отражений и переносов. Каждая точка пространства (кроме начальной точки О) может быть задана четверкой одновременно не равных нулю чисел (hx, hy,hz,h); эта четверка чисел определена однозначно с точностью до общего множителя.

Рассмотрим матрицы преобразований.

1) Матрицы вращения в пространстве

Матрица вращения вокруг оси абсцисс на угол Х

Матрица вращения вокруг оси ординат на угол Х

Матрица вращения вокруг оси аппликат на угол Х

2) Матрица растяжения (сжатия):

где

?>0 – коэффициент растяжения (сжатия) вдоль оси абсцисс;

?>0 – коэффициент растяжения (сжатия) вдоль оси ординат;

?>0 – коэффициент растяжения (сжатия) вдоль оси аппликат.

3) Матрицы отражения.

Матрица отражения относительно плоскости xy:

Матрица отражения относительно плоскости yz:

Матрица отражения относительно плоскости zx:

4) Матрица переноса

где (?, ?, ?) - вектор переноса.

В OреnG1 существуют стандартные команды для задания ортографической (параллельной) и перспективной проекций. Первый тип проекции может быть задан командами void g1Ortho(G1double left, G1double right, G1double bottom, G1double top, G1double near, G1double far) и void g1uOrtho2D(G1double left, G1double right, G1double bottom, G1double top).

Первая команда создает матрицу проекции в усеченный объем видимости (параллелепипед видимости) в левосторонней системе координат. Параметры команды задают точки (left, bоttom, znеаr) и (right, tор, zfаr), которые отвечают левому нижнему и правому верхнему углам окна вывода. Параметры «near» и «fаr» задают расстояние до ближней и дальней плоскостей отсечения по удалению от точки (0,0,0) и могут быть отрицательными.

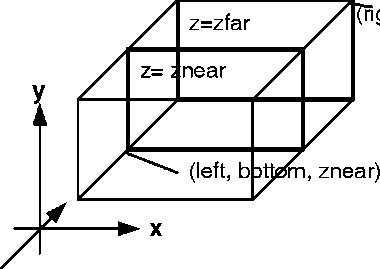


Рисунок 1 – Орто-проекция

Перспективная проекция определяется командой void g1uPerspective(G1double ang1ey, G1double aspect, G1double znear, G1doub1e zfar), которая задает усеченный конус видимости в левосторонней системе координат. Параметр «ang1ey» определяет угол видимости в градусах по оси у и должен находиться в диапазоне от 0 до 180. Угол видимости вдоль оси x задается параметром «aspect», который обычно задается как отношение сторон области вывода (как правило, размеров окна). Параметры «zfar» и «znear» задают расстояние от наблюдателя до плоскостей отсечения по глубине и должны быть положительными. Чем больше отношение zfar znear, тем хуже в буфере глубины будут различаться расположенные рядом поверхности, так как по умолчанию в него будет записываться «сжатая» глубина в диапазоне от 0 до 1.

Прежде чем задавать матрицы проекций, нужно включить режим работы с нужной матрицей командой g1MatrixMode(G1\_PROJECTION) и сбросить текущую, вызвав g1 Load Identity ( ) .

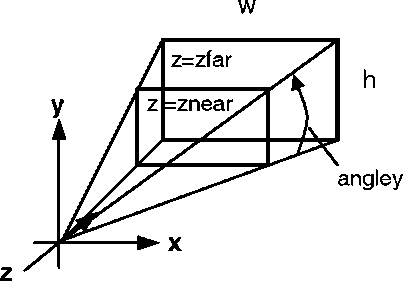


Рисунок 2 –Проекция-перспектива

## **Модель освещения(G1\_LIGTHING)**

По умолчанию освещение отключено. Включается оно командой g1Enable(G1\_LIGHTING).В OреnG1 используется модель освещения, в соответствии с которой цвет точки определяется несколькими факторами: свойствами материала и текстуры, величиной нормали в этой точке, а также положением источника света и наблюдателя. Для корректного расчета освещенности в точке надо использовать единичные нормали, однако команды: типа g1Scale\*(), могут изменять длину нормалей. Чтобы это учитывать, нужно использовать режим нормализации векторов нормалей, который включается вызовом команды g1Enable(G1\_NORMALIZE) .

Для задания глобальных параметров освещения используются команды void g1LightModel[i, f](G1enum pname, G1enum param) и void g1LightModel[i f]v(G1enum pname, const G1type \*params).

Аргумент «pname» определяет, какой параметр модели освещения будет настраиваться и может принимать следующие значения: G1\_LIGHT\_MODEL\_LOCAL\_VIEWER, параметр «param» должен быть булевым и задает положение наблюдателя. Если он равен G1\_FALSE, то направление обзора считается параллельным оси z, вне зависимости от положения в видовых координатах. Если же он равен G1\_TRUE, то наблюдатель находится в начале видовой системы координат. Это может улучшить качество освещения, но усложняет его расчет. Значение по умолчанию – G1\_FALSE.

G1\_LIGHT\_MODEL\_TWO\_SIDEпараметр «param» должен быть булевым и управляет режимом расчета освещенности, как для лицевых, так и для обратных граней. Если он равен G1\_FALSE, то освещенность рассчитывается только для лицевых граней. Если же он равен G1\_TRUE, расчет проводится и для обратных граней. Значение по умолчанию – G1\_FALSE.

G1\_LIGHT\_MODEL\_AMBIENT параметр «params» должен содержать четыре целых или вещественных числа, которые определяют цвет фонового освещения даже в случае отсутствия определенных источников света. Значение по умолчанию – (0.2, 0.2, 0.2,1.0).

## **1.5 Материалы**

Для задания параметров текущего материала используются команды void g1Material[i f](G1enum face, G1enum pname, G1type param) void g1Material[i f]v(G1enum face, G1enum pname, G1type \*params).

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

* G1\_AMBIENTпараметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют рассеянный цвет материала (цвет материала в тени). Значение по умолчанию – (О.2, О.2, О.2, 1.О);
* G1\_DIFFUSEпараметр «params» должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют диффузный цвет материала. Значение по умолчанию – (О.8, О.8, О.8, 1.О);
* G1\_SPECULAR параметр «params» должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют зеркальный цвет материала. Значение по умолчанию – (О.О, О.О, О.О, 1.О);
* G1\_SHININESS параметр params должен содержать одно целое или вещественное значение в диапазоне от 0 до 128, которое определяет степень зеркального отражения материала. Значение по умолчанию – О;
* G1\_EMISSION параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют интенсивность излучаемого света материала. Значение по умолчанию: (О.О, О.О, О.О, 1.О);
* G1\_AMBIENT\_AND\_DIFFUSE эквивалентно двум вызовам команды: g1Material\*() со значением «pname» G1\_AMBIENT и G1\_DIFFUSE и одинаковыми значениями «params».

Из этого следует, что вызов команды: g1Material[i f]() возможен только для установки степени зеркального отражения материала. Команда g1Material[i f]v() используется для задания остальных параметров.

Параметр «face» определяет тип граней, для которых задается этот материал и может принимать значения G1\_FRONT, G1\_BACK или G1\_FRONT\_AND\_BACK.

Если в сцене материалы объектов различаются лишь одним параметром, рекомендуется сначала установить нужный режим, вызвав g1Enable() с параметром G1\_COLOR\_MATERIAL, а затем использовать команду void g1ColorMaterial(G1enum face, G1enum pname), где параметр «face» имеет аналогичный смысл, а параметр «pname» может принимать все перечисленные значения. После этого значения выбранного с помощью «pname» свойства материала для конкретного объекта (или вершины) устанавливаются вызовом команды g1Color\*(), что позволяет избежать вызовов более ресурсоемкой команды g1Material\*() и повышает эффективность программы.

# **2. Разработка приложения для построения динамической изображения трехмерной модели объекта «Паровоз»**

## **2.1 Разработка процедуры визуализации трехмерной сцены**

За прорисовку паровоза отсвечает метод void CLr1View::OnDraw(CDC\* pDC). Вся модель состоит из довольно большого числа примитивов, таких как цилиндр, конус, прямоугольник, сфера.

Первым рисуем сферу (в начале координат), после нее –цилиндр, затем этим фигуры совмещаем нужным образом, делаем это при помощи функции «g1Translated». За тем рисуем прямоугольник, который пододвигаем к концу цилиндра и развернуть на 90 градусов относительно оси х (g1Translated, g1Rotated). Дальше изображаем еще два одинаковых цилиндра, которые отличаются от первого только радиусом. Они также совмещены с остальными деталями объекта при помощи функций g1Translated, g1Rotated. За тем рисуем колеса паровоза которые соединяем с цилиндрами, сначала четыре правых за тем четыре левых, так же с помощью функций g1Translated, g1Rotated. После этого, в нужном месте (опять при помощи g1Translated, g1Rotated), изображаем конус, который поворачиваем относительно оси OX и перемещаем его в нужное нам место.

Все необходимые операции производились теми же функциями g1Translated, g1Rotated.

Вот собственно и вся модель. Весь код метода представлен в Приложении А, а алгоритм построения трехмерного объекта «Паровоз» средствами OРЕNG1, в Приложении В.

Так же в сцене присутствуют источник света. Можно включать/выключать эффект прозрачности, отражения и камеры, помимо этого можно менять направление движения паровоза. В прочем, теория описана в первой части проекта, а код – в приложении А.

## **2.2 Разработка интерфейса пользователя**

Через интерфейс, пользователь может изменять работу программы. Имеются следующие возможности:

* настраивать проекции;
* включать/выключать источник света;
* включать/выключать эффект прозрачности;
* включать/выключать эффект отражения;
* включать/выключать камеры;
* разворачивать объект относительно дух осей;

Для добавления этих возможностей в приложении отредактировано главное меню. А именно добавлены следующие пункты (и обработчики их выбора):

* Right – повернуть объект вправо;
* Left – повернуть объект влево;
* Up – повернуть объект вверх;
* Down – повернуть объект вниз;
* Light 0n/Off – включить или выключить первый источник света;

Вызов каждого пункта меню можно заменить нажатием на клавиши, таблица соответствия приведена ниже.

Таблица 1 – «Горячие» клавиши

|  |  |
| --- | --- |
| Пункт меню | «Горячая» клавиша |
| вправо | Right |
| влево | Left |
| вверх | Up |
| вниз | Down |
| Включить освещение | L |
| Выключить освещение | K |
| Камера по оси X | W |
| Камера по оси X | S |
| Камера по оси Y | Q |
| Камера по оси Y | E |
| Камера по оси Z | A |
| Камера по оси Z | D |
| Движение вперёд/назад | N |
| Включение прозрачности | J |
| Выключение прозрачности | H |

## **2.3 Разработка подсистемы управлениями событиями**

Любое windows-положение основано на событиях и их обработке, другими словами поведение программы управляется событиями. Данный проект тоже является windows приложением, и, следовательно, обработка событий занимает важное место. К основным событиям, играющим важную, для корректной работы программы, роль относятся следующие:

* WM\_DESTROY - освобождение занятых ресурсов;
* WM\_SIZE - изменения сцены относительно размеров окна;
* WM\_ERASEBKGND - предотвращения мерцания;
* WM\_TIMER – используется для создания таймера;
* WM\_KEYDOWN – обработка нажатия клавиши;
* ON\_WM\_CREATE()
* ON\_WM\_TIMER()
* ON\_COMMAND(ID\_FILE\_PRINT, CView::OnFilePrint)
* ON\_COMMAND(ID\_FILE\_PRINT\_DIRECT, CView::OnFilePrint)
* ON\_COMMAND(ID\_FILE\_PRINT\_PREVIEW, CView::OnFilePrintPreview)

В обработчике события «WM\_SIZE» выполняется код, необходимый для изменения сцены относительно размеров окна, в обработчике события «WM\_DESTROY» освобождение занятых ресурсов. В обработчик «WM\_ERASEBKGND» вставлен код для предотвращения мерцания. Остальные события служат для обработки событий выбор пунктов главного меню. Из описании приведенных выше хорошо видно что для чего служит (см. Приложение 4).

# **3. Информационное и программное обеспечение**

## **3.1 Общие сведения о программе**

Программа называется «Трехмерная модель Паровоз». При её использовании, пользователь получает возможность работать с визуальной моделью паровоза. Вращать её (модель) относительно двух осей, изменять освещение. Включать/выключать эффект прозрачности. Программное обеспечение, на котором разработана приложение – Microsoft Visual Studio6.0.

## **3.2 Функциональное назначение**

Данная программа предназначается для представления трехмерной модели паровоза. Приложение дает следующие возможности:

* рассматривать 3d-модель;
* изменять её положение в пространстве (вращать относительно двух осей);
* включать камеры по осям;
* настраивать переспективу;
* включать или выключать эффект прозрачности;
* включать или выключать освещение;
* выбирать типы проекции;

## **3.3 Логическая структура и функциональная декомпозиция проекта**

Во-первых происходит инициализация OреnG1, она происходит в два этапа.

oреng1 программирование проекция визуализация

Первый этап. В методе «PreCreateWindow» устанавливается стиль окна, после чего – формат пикселей, это делается функцией usr\_bSetupPixelFormat, в котором в свою очередь выполняются ChoosePixelFormat и SetPixelFormat, которые проверяют формат и устанавливают его соответственно.

Второй этап. Создается контекст воспроизведения, за это отвечает wg1CreateContext, эта функция создает новый контекст воспроизведения OреnG1, который совместим с устройством, предоставленным дескриптором hdc. За этим устанавливается текущий контекст wg1MakeCurrent (для удаления контекста есть функция wg1DeleteContext. За инициализацию отвечает метод usr\_bInitOреnG1.

При завершении работы программы нужно освободить занятые ресурсы, за это отвечает метод user\_DestoryOреnG1.

Еще один важный метод (usr\_ReSize) устанавливает область вывода, и перспективу. Он вызывается при каждом изменении размера окна.

Рисование самой сцены. За это отвечает метод usr\_RenderScene, который вызывается при каждой перерисовки окна, и в свою очередь вызывает вспомогательные функции рисования.

Так как данная программа является так называемым windows-приложением, то алгоритм её работы зависит от конкретных событий, а пользователь, в свою очередь, получает возможность, через интерфейс, «порождать» эти события. Поэтому важную роль играют обработчики событий, таких как выбор пунктов меню, нажатие горячих клавиш (которые собственно синхронизированы между собой). Самые важные для понимания логической структуры программы события описаны в разделе «2.3 Разработка подсистемы управления событиями». В итоге приведена таблица, в которой отображены основные методы (функции) и описано, за что они отвечают.

Таблица 2 – Основные методы и функции

|  |  |
| --- | --- |
| Имя функции (метода) | Описание |
| PreCreateWindow | Инициализация окна |
| usr\_bSetupPixelFormat | Установка формата пикселей |
| usr\_bInitOреnG1 | Инициализация OреnG1 |
| user\_DestoryOреnG1 | Освобождение ресурсов (из-под OреnG1) |
| usr\_ReSize | Корректирует вывод сцены на экран при изменении размера окна |
| usr\_RenderScene | Рисует всю сцену (в частности вызывает методы рисования) |
| Axis | Вспомогательная функция, рисует оси координат |

## 

## **3.4 Требования к техническому программному обеспечению**

- процессор 1 ГГц (32- или 64-битный);

- 1 Гб RAM для 32-битной системы и 2 Гб RAM для 64-битной;

- 16 Гб свободного пространства на диске для 32-битной системы и 20 Гб свободного пространства на диске для 64-битной;

- Видеокарта поддержкой DirectX 9 с драйвером WDDM 1.0 или выше.

Работоспособность программы проверялась в системе Windows 7 32 bit при разрешении экрана 190\*1200.

## **3.5 Руководство пользователя**

Данная программа дает возможность пользователю увидеть трехмерную (3d) модель паровоза, а также:

* включать / выключать эффект прозрачности;
* настраивать положение модели в пространстве (вращение вокруг двух осей);
* включать или же выключать освещение;
* менять направление движения вращения паровоза;

Чтобы изменить положение паровоза в пространстве нужно выполнить одно из следующих действий:

* нажить на одну из кнопок «налево» «направо» «вверх» «вниз» для поворота в соответствующую сторону.

Чтобы включить/выключить эффект прозрачности нужно

* нажать на клавишу «J», для включения прозрачности
* нажать на клавишу «H, для выключения прозрачности».

Чтобы включить/выключить освещение нужно:

* выбрать пункт меню «Light» → «Light on» «Light off»;
* нажать на одну из клавиш «L» - для включения освещения и «K» - для выключения освещения.

Чтобы изменить направление движения паровоза нужно нажать на клавишу «N».

# Для запуска приложения необходимо всего лишь скопировать его исполняемый файл в удобную папку и запустить, после чего можно начать работу с приложением, которое имеет интуитивно – понятный интерфейс.

# **Заключение и выводы**

В результате проделанной работы, разработали приложение с использованием библиотеки OреnG1 для построения динамического изображения трехмерной модели объекта «Паровоз».

При использовании данной модели, пользователь получает возможность работать с визуальной моделью паровоза, т.е.:

– вращать модель относительно двух осей;

– включать/выключать освещение;

– включать/выключать эффект прозрачности;

– менять направление движения паровоза;

– настраивать камеру относительно оси Х;

– настраивать камеру относительно оси Y;

– настраивать камеру относительно оси Z.

# **Список литературы**

* 1. Порев В.Н. Компьютерная графика. СПб., BHV, 2002.
  2. Шикин А.В., Боресков А.В. Компьютерная графика. Полигональные модели. Москва, ДИАЛОГ-МИФИ, 2001.
  3. Тихомиров Ю. Программирование трехмерной графики. СПб,
  4. BHV, 1998.
  5. 9.Performance OреnG1: Platform Independent Techniques.
  6. SIGGRAPH 2001 course.
  7. OреnG1 performance optimization, Siggraph'97 course.
  8. Visual Introduction in OреnG1, SIGGRAPH'98.
  9. The OреnG1 graphics system: a specification (version 1.1).
  10. Программирование G1UT: окна и анимация. Miguel Angel Sepulveda, LinuxFocus.
  11. The OреnG1 Utility Toolkit (G1UT) Programming Interface, API version 3, specification.

# 

# **Приложение 1**

# 

Рисунок 3 – Диаграмма классов

# **Приложение 2**

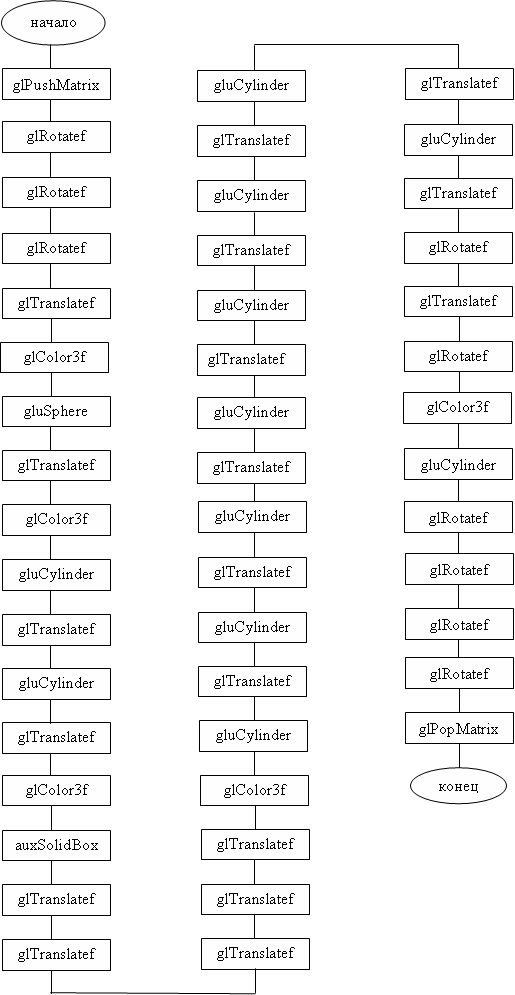
****

Рисунок 4 – Алгоритм построения объекта «паровоз»

# **Приложение 3**

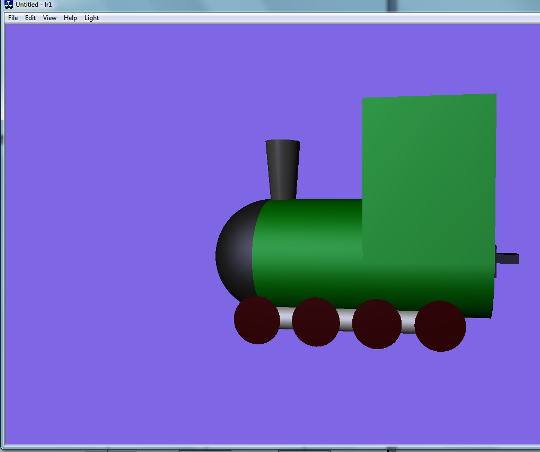


Рисунок 5 – Трехмерная модель объекта «Паровоз»

# **Приложение 4**



Рисунок 6 – Прототип модели

# **Приложение 5**

**Листинг программы**

// lr1View.cpp : implementation of the CLr1View class

//

#include "stdafx.h"

#include "lr1.h"

#include "lr1Doc.h"

#include "lr1View.h"

#include "math.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#undef THIS\_FILE

static char THIS\_FILE[] = \_\_FILE\_\_;

#endif

/////////////////////////////////////////////////////////////////////////////

// CLr1View

IMPLEMENT\_DYNCREATE(CLr1View, CView)

BEGIN\_MESSAGE\_MAP(CLr1View, CView)

//{{AFX\_MSG\_MAP(CLr1View)

ON\_WM\_CREATE()

ON\_WM\_DESTROY()

ON\_WM\_SIZE()

ON\_WM\_ERASEBKGND()

ON\_WM\_KEYDOWN()

ON\_WM\_TIMER()

//}}AFX\_MSG\_MAP

// Standard printing commands

ON\_COMMAND(ID\_FILE\_PRINT, CView::OnFilePrint)

ON\_COMMAND(ID\_FILE\_PRINT\_DIRECT, CView::OnFilePrint)

ON\_COMMAND(ID\_FILE\_PRINT\_PREVIEW, CView::OnFilePrintPreview)

END\_MESSAGE\_MAP()

/////////////////////////////////////////////////////////////////////////////

// CLr1View construction/destruction

CLr1View::CLr1View()

{

// TODO: add construction code here

}

CLr1View::~CLr1View()

{

}

BOOL CLr1View::PreCreateWindow(CREATESTRUCT& cs)

{

// TODO: Modify the Window class or styles here by modifying

// the CREATESTRUCT cs

cs.style|=(WS\_CLIPCHILDREN|WS\_CLIPSIBLINGS);

return CView::PreCreateWindow(cs);

}

//BOOL CLr1View::usr\_bsetup

/////////////////////////////////////////////////////////////////////////////

// CLr1View drawing

void CLr1View::OnDraw(CDC\* pDC)

{

CLr1Doc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

usr\_RenderScene();

// TODO: add draw code for native data here

}

/////////////////////////////////////////////////////////////////////////////

// CLr1View printing

BOOL CLr1View::OnPreparePrinting(CPrintInfo\* pInfo)

{

// default preparation

return DoPreparePrinting(pInfo);

}

void CLr1View::OnBeginPrinting(CDC\* /\*pDC\*/, CPrintInfo\* /\*pInfo\*/)

{

// TODO: add extra initialization before printing

}

void CLr1View::OnEndPrinting(CDC\* /\*pDC\*/, CPrintInfo\* /\*pInfo\*/)

{

// TODO: add cleanup after printing

}

/////////////////////////////////////////////////////////////////////////////

// CLr1View diagnostics

#ifdef \_DEBUG

void CLr1View::AssertValid() const

{

CView::AssertValid();

}

void CLr1View::Dump(CDumpContext& dc) const

{

CView::Dump(dc);

}

CLr1Doc\* CLr1View::GetDocument() // non-debug version is inline

{

ASSERT(m\_pDocument->IsKindOf(RUNTIME\_CLASS(CLr1Doc)));

return (CLr1Doc\*)m\_pDocument;

}

#endif //\_DEBUG

/////////////////////////////////////////////////////////////////////////////

// CLr1View message handlers

BOOL CLr1View::usr\_bSetupPixelFormat()

{

int iPixelType = PFD\_TYPE\_RGBA;

DWORD dwFlags = PFD\_DOUBLEBUFFER|

PFD\_SUPPORT\_OРЕNG1|

PFD\_DRAW\_TO\_WINDOW;

PIXELFORMATDESCRIPTOR pfd = {0};

pfd.nSize = sizeof(PIXELFORMATDESCRIPTOR);

pfd.nVersion = 1;

pfd.dwFlags = dwFlags;

pfd.iPixelType = iPixelType;

pfd.cColorBits =64;

pfd.cAlphaBits =64;

pfd.cAccumBits =64;

pfd.cDepthBits =64;

pfd.cStencilBits =64;

pfd.cAuxBuffers =64;

pfd.iLayerType = PFD\_MAIN\_PLANE;

CClientDC\*lDC=new CClientDC(this);

int pixelformat;

if ((pixelformat = ChoosePixelFormat(lDC->GetSafeHdc(), &pfd))==0)

{

MessageBox ("usr\_bSetupPixelFormat:ChoosePixelFormat failed");

lDC->DeleteDC();

return FALSE;

}

if (SetPixelFormat(lDC->GetSafeHdc(), pixelformat, &pfd)==FALSE)

{

MessageBox ("usr\_bSetupPixelFormat:SetPixelFormat failed");

lDC->DeleteDC();

return FALSE;

}

lDC->DeleteDC();

return TRUE;

}

BOOL CLr1View::usr\_bInitOреnG1()

{

HG1RC hrc;

tmpDC = new CClientDC(this);

if (!usr\_bSetupPixelFormat()) return FALSE;

hrc = wg1CreateContext(tmpDC->GetSafeHdc());

if (!wg1MakeCurrent(tmpDC->GetSafeHdc(), hrc)) return FALSE;

usr\_PreInit();

return TRUE;

}

void CLr1View::usr\_DestroyOреnG1()

{

HG1RC hrc;

hrc=::wg1GetCurrentContext();

::wg1MakeCurrent(NULL, NULL);

if (hrc)

::wg1DeleteContext(hrc);

if (tmpDC)

tmpDC->DeleteDC();

}

int CLr1View::OnCreate(LPCREATESTRUCT lpCreateStruct)

{

if (CView::OnCreate(lpCreateStruct) == -1)

return -1;

if (!this->usr\_bInitOреnG1())

{

AfxMessageBox("ERROR!!!");

return -1;

}

return 0;

}

void CLr1View::OnDestroy()

{

CView::OnDestroy();

this->usr\_DestroyOреnG1();

}

void CLr1View::OnSize(UINT nType, int cx, int cy)

{

CView::OnSize(nType, cx, cy);

usr\_ReSize(0, 0, cx, cy);

}

void CLr1View::usr\_ReSize(int x, int y, int width, int height)

{

if (height==0)height=100; // óñòðàíåíèå âîçìîæíîé îøèáêè ñ èíèöèàëèçàöèåé âûñîòû îêíà

G1float stor=(G1float)(((G1float)width)/((G1float)height)); // ñ÷èòàåì ñîîòíîøåíèå ñòîðîí (äëÿ ìàñøòàáèðîâàíèÿ)

g1Viewport (0,0,(G1int)width,(G1int)height); // ñîçäà¸ì îáëàñòü äëÿ âûâîäà èçîáðàæåíèÿ

g1MatrixMode(G1\_PROJECTION); // óñòàíàâëèâàåì òèï ìàòðèöû äëÿ ïðîåêöèè

g1LoadIdentity(); // çàãðóæàåì åäèíè÷íóþ ìàòðèöó

g1uPerspective(45.0f,(G1float)width/(G1float)height,0.1f,100.0f); // ðåæèì ðàáîòû ñ ìàòðèöàìè - ïåðñïåêòèâà.

g1MatrixMode(G1\_MODELVIEW); // ðåæèì ìàòðèöû - ïðîñìîòð ìîäåëè

}

BOOL CLr1View::OnEraseBkgnd(CDC\* pDC)

{

return TRUE;

}

void CLr1View::usr\_RenderScene()

{

g1ClearColor(0.5f, 0.4f, 0.9f, 1.0f); // óñòàíîâêà ôîíà

g1Clear(G1\_COLOR\_BUFFER\_BIT| G1\_DEPTH\_BUFFER\_BIT); // î÷èñòêà

g1LoadIdentity(); // çàãðóçêà åäèíè÷íîé ìàòðèöû

g1uLookAt(x\_e+2,y\_e+0.8,z\_e+8,0,0,0.1f\*1-8,x\_e+2,y\_e+5.3,z\_e+8-0.1f\*1+8); // ïîëîæåíèå è íàïðàâëåíèå êàìåðû

g1Materialf(G1\_FRONT,G1\_SHININESS,10); // ñèëà îòðàæåíèÿ 1 èç 128

g1Rotatef(t\_t\*15, 0, 1,0);

//g1Translatef(0.0f,-2.2f,0.0f); // ïåðåìåùàåì "3D êóðñîð"

g1Color4f(0.5f,0.5f,0.6f,1.0f); // óñòàíàâëèâàåò öâåò îáüáåêòà

//g1Translatef(0.0f,2.2f,0.0f); // ïåðåìåùàåìñÿ

g1Rotatef(y\_rotate, 0, 1,0);

g1Rotatef(x\_rotate, 1, 0,0);

g1Translatef(0.0f,0.0f,-1.0f);

auxSolidSphere(1.0); // ðèñóåì ñôåðó

g1Color3f(0.1f, 0.1f, 0.1f);

g1uSphere(q, 1.0f, 64, 32);

g1Translatef(0.0f,0.0f,0.0f);

g1Color3f(0.0f, 0.35f, 0.0f);

g1uCylinder(q, 1.0f,1.0f, 3.8f, 64, 32);//öèëëèíäð

g1Translatef(0.0f,0.0f,3.8f);

g1uCylinder(q, 1.0f,0.0f, 0.0f, 64, 32);//öèëëèíäð-çàãëóøêà

g1Translatef(0.0f,1.25f,-1.0f);

g1Translatef(0.0f,0.0f,-0.1f); //îêíî

g1Materialf(G1\_FRONT,G1\_SHININESS,0);

g1Color3f(0.55f, 0.55f, 0.55f);

auxSolidBox(1.5f,1.1f,1.85f);

g1Translatef(0.0f,0.0f,0.1f);

g1Materialf(G1\_FRONT,G1\_SHININESS,10);

g1Color3f(0.0f, 0.35f, 0.0f);

auxSolidBox(2.0f,2.5f,2.0f);//êàáèíà

g1Translatef(0.0f,-1.25f,1.0f);

g1Color3f(0.0f, 0.0, 0.0f);

auxSolidBox(1.4f,0.5f,0.1f);//øòóêà äëÿ âàãîíîâ

g1Color3f(0.0f, 0.0, 0.0f);

auxSolidBox(0.3f,0.15f,0.9f);//øòóêà äëÿ âàãîíîâ 2

g1Translatef(0.0f,0.0f,0.8f);

g1Translatef(0.0f,0.0f,0.4f);

g1Translatef(0.0f,0.0f,-0.01f);

g1Translatef(0.5f,-1.0f,-5.0f);

//îñè

g1Color3f(0.5f, 0.5f, 0.5f);

g1uCylinder(q, 0.2, 0.2f, 3.0f, 64, 32);

g1Translatef(-1.0f,0.0f,0.0f);

g1uCylinder(q, 0.2, 0.2f, 3.0f, 64, 32);

g1Translatef(-0.30f,0.0f,0.0f);

g1Rotatef(90, 0, 1,0);

g1Translatef(0.0f,0.0f,0.0f);

// êîëåñà

g1Materialf(G1\_FRONT,G1\_SHININESS,60);

g1Color3f(0.15f, 0.0f, 0.0f);

g1uCylinder(q, 0.0f, 0.4f, 0.0f, 64, 32);

g1Translatef(-1.0f,0.0f,0.0f);

g1uCylinder(q, 0.0f, 0.4f, 0.0f, 64, 32);

g1Translatef(-1.0f,0.0f,0.0f);

g1uCylinder(q, 0.0f, 0.4f, 0.0f, 64, 32);

g1Translatef(-1.0f,0.0f,0.0f);

g1uCylinder(q, 0.0f, 0.4f, 0.0f, 64, 32);

g1Translatef(0.0f,0.0f,1.6f);

g1uCylinder(q, 0.4f, 0.0f, 0.0f, 64, 32);

g1Translatef(1.0f,0.0f,0.0f);

g1uCylinder(q, 0.4f, 0.0f, 0.0f, 64, 32);

g1Translatef(1.0f,0.0f,0.0f);

g1uCylinder(q, 0.4f, 0.0f, 0.0f, 64, 32);

g1Translatef(1.0f,0.0f,0.0f);

g1uCylinder(q, 0.4f, 0.0f, 0.0f, 64, 32);

g1Translatef(0.0f,0.0f,-1.32f);

g1Rotatef(-90, 0, 1,0);

g1Translatef(0.5f,3.0f,0.2f);

g1Rotatef(90, 1, 0,0);

//òðóáà

g1Color3f(0.2f, 0.2f, 0.2f);

g1uCylinder(q, 0.3f, 0.1f, 2.5f, 64, 32);

g1Rotatef(-90, 1, 0,0);

g1Rotatef(t\_t\*15, 1, 0,0); // ïîâîðîò îòíîñèòåëüíî îñè îÕ

g1Rotatef(y\_rotate, 0, 1,0);

g1Rotatef(x\_rotate, 1, 0,0);

g1PopMatrix();

g1Finish();

SwapBuffers(::wg1GetCurrentDC());

}

void CLr1View::usr\_PreInit()

{

// èíèöèàëèçàöèÿ íà÷àëüíûõ ïîëîæåíèé è òàéìåðà

x\_rotate=0;

y\_rotate=0;

t\_t=0;

t\_n=0.22f;

x\_e=0;

y\_e=0;

z\_e=0.7f;

////////////

float mat\_spec[] ={0.2f,0.2f,0.3f}; // ýôôåêò îòðàæåíèÿ

float pos[3]={-0.8f,0.5f,0.0f}; // ïîçèöèÿ èñòî÷íèêà ñâåòà

float dir[3]={0.0f,0.0f,0.0f}; // íàïðàâëåíèå ñâåòà

g1Enable (G1\_DEPTH\_TEST); // âêëþ÷åíèå òåñòà ãëóáèíû (ïåðåêðûòèå îáüåêòîâ)

g1Enable (G1\_COLOR\_MATERIAL); // ðàçðåøåíèå öâåòíûõ îáüåêòîâ

g1Materialfv(G1\_FRONT,G1\_SPECULAR,mat\_spec); // ýôôåêò îòðàæåíèÿ

g1Materialf(G1\_FRONT,G1\_SHININESS,10); // ýôôåêò îòðàæåíèÿ

g1Enable (G1\_LIGHTING); // ðàçðåøåíèå îñâåùåíèÿ

g1Enable (G1\_LIGHT0); // âêëþ÷åíèå èñòî÷íèêà ñâåòà

g1Lightfv(G1\_LIGHT0,G1\_SPOT\_DIRECTION,dir); // íàïðàâëåíèå ñâåòà

g1Lightfv(G1\_LIGHT0,G1\_POSITION,pos); // ïîçèöèÿ èñòî÷íèêà ñâåòà

g1ShadeModel(G1\_SMOOTH); // âêëþ÷åíèå ñãëàæèâàíèÿ

g1MatrixMode(G1\_MODELVIEW); // âûáîð âèäà ìàòðèöû

q = g1uNewQuadric(); // Ñîçäàòü êâàäðàòè÷íûé îáúåêò

// òèï ãåíåðèðóåìûõ íîðìàëåé äëÿ íåãî – «ñãëàæåííûå»

g1uQuadricNormals(q, G1\_SMOOTH);

SetTimer(1, 50, NULL); // óñòàíîâêà òàéìåðà äëÿ ñ÷¸ò÷èêà

}

void CLr1View::usr\_DrawAxis()

{

g1Begin(G1\_LINES);

g1Color3f(1,0,0);

g1Vertex3f(0,0,0);

g1Vertex3f(2,0,0);

g1Color3f(0,1,0);

g1Vertex3f(0,0,0);

g1Vertex3f(0,2,0);

g1Color3f(0,0,1);

g1Vertex3f(0,0,0);

g1Vertex3f(0,0,2);

g1End();

}

void CLr1View::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)

{

switch(nChar)

{

case VK\_DOWN :x\_rotate += 3;break; // âíèç

case VK\_UP :x\_rotate -= 3;break; // ââåðõ

case VK\_LEFT :y\_rotate += 3;break; // âëåâî

case VK\_RIGHT :y\_rotate -= 3;break; // âïðàâî

case 87 :x\_e += 0.1f;break; // êàìåðà ïî õ W

case 83 :x\_e -= 0.1f;break; // êàìåðà ïî õ S

case 81 :y\_e += 0.1f;break; // êàìåðà ïî ó Q

case 69 :y\_e -= 0.1f;break; //êàìåðà ïî ó E

case 65 :z\_e += 0.1f;break; //êàìåðà ïî z A

case 68 :z\_e -= 0.1f;break; //êàìåðà ïî z D

case 78 :t\_n = -t\_n;break; //äâèæåíèå âïåð¸ä/íàçàä N

case 76 :g1Enable (G1\_LIGHTING);break; //âêë îñâåùåíèå L

case 75 :g1Disable (G1\_LIGHTING);break; //âûêë îñâåùåíèå K

case 74 :g1Enable(G1\_BLEND);g1Disable(G1\_DEPTH\_TEST); break; //âêë ïðîçðà÷íîñòè J

case 72 :g1Disable(G1\_BLEND);g1Enable(G1\_DEPTH\_TEST);break; //âûêë ïðîçðà÷íîñòè H

}

//Invalidate();// íå íóæåí, ò.ê. âêëþ÷åí òàéìåð

CView::OnKeyDown(nChar, nRepCnt, nFlags);

}

void CLr1View::OnTimer(UINT nIDEvent)

{

t\_t+=t\_n; // ñ÷¸ò÷èê òàéìåðà

Invalidate(); // ïåðåïðîðèñîâêà

CView::OnTimer(nIDEvent);

}

Размещено на Allbest.ru