Лабораторная работа № 4

Векторно-полигональная и аналитическая модели объекта. Преобразования в пространстве (3D-преобразования)

Цель работы – изучение и реализация алгоритмов построения и преобразования трехмерного объекта с использованием векторно-полигональной и аналитической моделей; изучение и реализация алгоритмов построения проекции фигуры, удаления невидимых линий и граней; изучение принципов применения библиотеки OpenGL при разработке приложений в C#.

Для выполнения лабораторной работы ознакомьтесь с лекционным материалом по рассматриваемой теме.

Задания к лабораторной работе № 4

– ***Задание 1***

1. Создайте проект, в котором реализуйте процедуры преобразований многогранника в пространстве. Отображаемую фигуру выберите в соответствии с вариантом (таблица 1). Описание фигур представлено в приложении Б.

2. Координаты вершин многогранника – мировые.

3. Отобразите многогранник так, чтобы высота фигуры совпадала с координатной осью Y. Для отображения многогранника используйте ортогональную проекцию.

4. При необходимости поверните многогранник на экране так, чтобы было видно его форму.

– ***Задание 2***

Нарисуйте систему координат.

– ***Задание 3***

В программном модуле необходимо предусмотреть следующие возможности:

– перемещение многогранника (по каждой координатной оси, по двум или трем осям одновременно);
– отображение относительно координатных плоскостей;
– изменение размера многогранника (по каждой оси, по двум или трем координатным осям одновременно);
– вращение вокруг координатных осей;
– изменение направления и скорости вращения / перемещения.

Таблица 1 – Варианты заданий лабораторной работы 4

№ вар.	Фигура	№ вар.	Фигура
1.	Тетраэдр	2.	Шестигранник, образованный треугольниками
3.	Тетраэдр	4.	Шестигранник, образованный треугольниками
5.	Тетраэдр	6.	Шестигранник, образованный треугольниками
7.	Тетраэдр	8.	Шестигранник, образованный треугольниками
9.	Гексаэдр (куб)	10.	Гексаэдр (куб)
11.	Гексаэдр (куб)	12.	Гексаэдр (куб)

Задания для самостоятельного выполнения к лабораторной работе № 4

– ***Общие требования к программному продукту и защите проекта***

1. Максимальное и минимальное значение мировых координат изобразите на экране.

2. 21.04.1962 был создан ТИРЭТ (в настоящее время ТУСУР). Отобразите в проекте фразу «ТУСУР - XXX года. ТУСУР – Чемпион!» и логотип ТУСУР.

3. При оформлении отчета по индивидуальной части лабораторной работы придумайте по два тестовых вопроса по изученным и реализованным алгоритмам. В каждом отчете должны быть представлены вопросы разного типа (закрытые вопросы с выбором одного правильного ответа, закрытые вопросы с выбором нескольких правильных ответов, открытый вопрос с вводом

ответа в виде числа или строки, вопрос на установление соответствия, вопрос-эссе, вопрос на установление правильной последовательности и т.д.).

– ***Задание для руководителя***

1. Откройте проект, созданный в лабораторной работе №1. Для кнопки «Лабораторная работа №4» создайте обработчик (модуль), демонстрирующий все задания участников группы.

2. Реализуйте настройку цвета и стиля линии – толстая (предусмотрите выбор толщины и вида пера), тонкая, сплошная, пунктирная линия (предусмотрите настройку шага пунктира).

3. Реализуйте индивидуальное задание (номер варианта необходимо получить у преподавателя).

– ***Задание для технического писателя***

1. Реализуйте индивидуальное задание (номер варианта необходимо получить у преподавателя).

2. Напишите отчет к лабораторной работе. Требования к отчету см.п. «Методические указания по выполнению и защите лабораторных работ. Требования к оформлению отчета».

– ***Задание для остальных участников группы***

1. Реализуйте индивидуальное задание (номера вариантов необходимо получить у преподавателя).

Варианты индивидуальных заданий на самостоятельную работу к лабораторной работе № 4

1. Реализуйте построение на экране графика поверхности - трехмерной функции и разные варианты преобразования данного графика – вращение, перемещение и масштабирование. Вид функции выбирается по вариантам (табл. 2):

Таблица 2

№ вар.	Вид функции	Диапазон изменения x	Диапазон изменения y
1	$R = x^2 + y^2 \; Z = \frac{\sin(R)}{R}$	[-3; 3]	[-3; 3]
2	$Z = \sin^2(x) + \sin^2(y)$	[-3; 3]	[-3; 3]
3	$Z = (\sin(x) + \cos(y))^2$	[-3; 3]	[-3; 3]
4	$Z = \sin^2(x) + \cos(y)$	[-2; 2]	[-2; 2]
5	$Z = \sin(x) + \cos^2(y)$	[-2; 2]	[-2; 2]
6	$Z = x^2 - y^2 - 100$	[-3; 3]	[-3; 3]
7	$Z = e^{\sin(x) - y^2}$	[-3; 3]	[-3; 3]
8	$Z = e^{\sin(y) - x^2}$	[-3; 3]	[-3; 3]
9	$Z = e^{\cos(y) - x^2}$	[-3; 3]	[-3; 3]
10	$Z = e^{\sin(x) + y^2}$	[-3; 3]	[-3; 3]
11	$Z = e^{\sin(x) - \cos(y)}$	[-3; 3]	[-3; 3]
12	$Z = e^{\sin(x) + \cos(y)}$	[-3; 3]	[-3; 3]
13	$Z = e^{\sin(x) * \cos(y)}$	[-3; 3]	[-3; 3]
14	$Z = \cos(x^2 - y^2)$	[-2; 2]	[-2; 2]
15	$Z = \sin(x^2 - y^2)$	[-2; 2]	[-2; 2]

2. Напишите программный модуль для отображения результатов преобразований многогранника в пространстве с удалением невидимых частей. Для изображения многоугольника используйте каркасную модель. Невидимые ребра должны быть изображены пунктирными линиями.

2.1 Первым действием отобразите (поверните) многогранник на экране так, чтобы было видно его форму.

2.2 Реализуйте:

– построение оси вращения;

– поворот многогранника вокруг заданной оси вращения;

2.3 Нарисуйте систему координат и ось вращения.

2.4 Координаты точек – мировые.

2.5 Отображаемую фигуру, ось вращения, вид проекции и алгоритм удаления невидимых линий выберите в соответствии с вариантом (таблица 3):

Таблица 3

№	Фигура	Проекция	Ось вращения	Алгоритм удаления невидимых линий
1.	Октаэдр	Диметрия	Прямая под углом 45 градусов ко всем координатным осям и проходящая через начало координат	алгоритм Робертса
2.	Тетраэдр	Диметрия	Прямая под углом 45 градусов ко всем координатным осям и проходящая через начало координат	алгоритм, использующий нормаль грани
3.	Шестигранник, образованный треугольниками	Перспектива (2 точки схода)	Вертикальная прямая, параллельная оси OY. Смещение от оси OY задается пользователем интерактивно	алгоритм Робертса
4.	Тетраэдр	Перспектива (1 точка схода)	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм, использующий нормаль грани
5.	Октаэдр	Перспектива (1 точка схода)	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм Робертса
6.	Гексаэдр (куб)	Диметрия	Прямая под углом 45 градусов ко всем координатным осям и проходящая через начало координат	алгоритм, использующий нормаль грани
7.	Пятиугольная призма (карандаш)	Перспектива (1 точка схода)	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм Робертса
8.	Шестигранник, образованный треугольниками	Изометрия	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм, использующий нормаль грани

Продолжение таблицы 3

№	Фигура	Проекция	Ось вращения	Алгоритм удаления невидимых линий
9.	Октаэдр	Перспектива (2 точки схода)	Горизонтальная прямая, параллельная оси OX. Смещение от оси OX задается пользователем интерактивно	алгоритм Робертса
10.	Шестигранник, образованный треугольниками	Диметрия	Прямая под углом 45 градусов ко всем координатным осям и проходящая через начало координат	алгоритм, использующий нормаль грани
11.	Пятиугольная призма (карандаш)	Диметрия	Прямая под углом 45 градусов ко всем координатным осям и проходящая через начало координат	алгоритм Робертса
12.	Шестигранник, образованный треугольниками	Перспектива (1 точка схода)	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм, использующий нормаль грани
13.	Гексаэдр (куб)	Изометрия	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм Робертса
14.	Октаэдр	Изометрия	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм, использующий нормаль грани
15.	Гексаэдр (куб)	Перспектива (2 точки схода)	Вертикальная прямая, параллельная оси OY. Смещение от оси OY задается пользователем интерактивно	алгоритм Робертса
16.	Октаэдр	Перспектива (2 точки схода)	Горизонтальная прямая, параллельная оси OX. Смещение от оси OX задается пользователем интерактивно	алгоритм, использующий нормаль грани
17.	Пятиугольная призма (карандаш)	Изометрия	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм Робертса
18.	Гексаэдр (куб)	Изометрия	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм, использующий нормаль грани

Продолжение таблицы 3

№	Фигура	Проекция	Ось вращения	Алгоритм удаления невидимых линий
19.	Пятиугольная призма (карандаш)	Перспектива (2 точки схода)	Вертикальная прямая, параллельная оси OY. Смещение от оси OY задается пользователем интерактивно	алгоритм Робертса
20.	Гексаэдр (куб)	Перспектива (1 точка схода)	Прямая, заданная произвольными направляющими векторами и проходящая через начало координат	алгоритм, использующий нормаль грани
21.	Икосаэдр	Изометрия	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм Робертса
22.	Икосаэдр	Диметрия	Прямая под углом 45 градусов ко всем координатным осям и проходящая через начало координат	алгоритм, использующий нормаль грани
23.	Икосаэдр	Перспектива (2 точки схода)	Горизонтальная прямая, параллельная оси OX. Смещение от оси OX задается пользователем интерактивно	алгоритм Робертса
24.	Икосаэдр	Перспектива (1 точка схода)	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм, использующий нормаль грани

3. Реализуйте приемы работы с объектами, используя библиотеку OpenGL.

3.1 Изучите принципы применения библиотеки OpenGL при разработке приложений в C# (см. приложение В). Реализуйте представленный пример.

3.2 Модифицируйте пример так, чтобы происходило непрерывный поворот выбранного объекта.

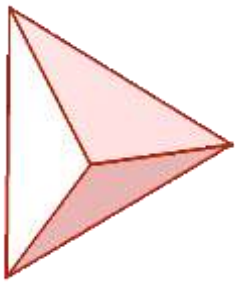
3.3 Реализуйте возможность изменения цвета объекта и перемещения точки освещения.

Контрольные вопросы к лабораторной работе № 4

1. Что такое векторно-полигональная модель объекта?
2. Опишите способы представления векторной полигональной модели
3. Опишите матрицу преобразований в пространстве общего вида. Раскройте назначение каждого элемента.
4. Опишите матрицы, используемые для 3D-преобразования.
5. Опишите процесс получения 3D-преобразований.
6. Что такое «аналитическая модель»?
7. Опишите процедуру отображения графика трехмерной функции на экране.
8. Опишите процедуры получения вращения, масштабирования и перемещения полученной поверхности.
9. Что такое проецирование? Для чего оно используется.
10. Опишите существующую классификацию видов проекций.
11. В чем основное различие параллельной и перспективной проекции.
12. Опишите матрицы проецирования.
13. Опишите процедуру получения вращения трехмерной фигуры на экране.
14. Сущность и назначение OpenGL.
15. Назначение Tao Framework.
16. Инициализация OpenGL в C#.
17. Опишите основные методы преобразования объектов, реализованные в OpenGL, использованные в лабораторной работе.

Приложение Б

Основные геометрические фигуры

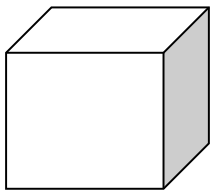


Тетраэдр

Тетраэдр составлен из четырех равносторонних треугольников. Каждая его вершина является вершиной трех треугольников. Сумма плоских углов при каждой вершине равна 180 градусов. Таким образом, тетраэдр имеет 4 грани, 4 вершины и 6 ребер.

Элементы симметрии:

Тетраэдр не имеет центра симметрии, но имеет 3 оси симметрии и 6 плоскостей симметрии.

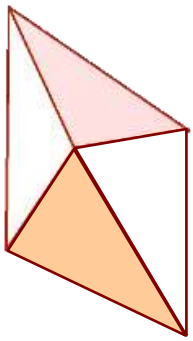


Гексаэдр (Куб)

Куб составлен из шести квадратов. Каждая его вершина является вершиной трех квадратов. Сумма плоских углов при каждой вершине равна 270 градусов. Таким образом, куб имеет 6 граней, 8 вершин и 12 ребер.

Элементы симметрии:

Куб имеет центр симметрии - центр куба, 9 осей симметрии и 9 плоскостей симметрии.

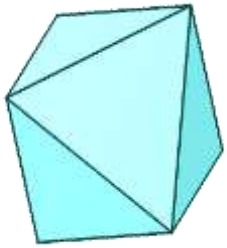


Шестигранник, образованный равносторонними треугольниками

Составлен из шести равносторонних треугольников. Две противоположные осевые вершины являются вершинами трех треугольников.

Элементы симметрии: имеет 4 оси симметрии.

Октаэдр

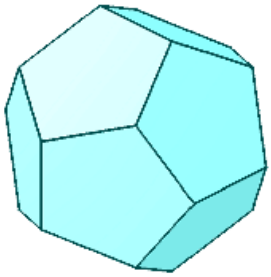


Октаэдр составлен из восьми равносторонних треугольников. Каждая его вершина является вершиной четырех треугольников. Сумма плоских углов при каждой вершине равна 240 градусов. Таким образом, октаэдр имеет 8 граней, 6 вершин и 12 ребер.

Элементы симметрии:

Октаэдр имеет центр симметрии - центр октаэдра, 9 осей симметрии и 9 плоскостей симметрии.

Додекаэдр



Додекаэдр составлен из двенадцати равносторонних пятиугольников. Каждая его вершина является вершиной трех пятиугольников. Сумма плоских углов при каждой вершине равна 324 градусов. Таким образом, додекаэдр имеет 12 граней, 20 вершин и 30 ребер.

Элементы симметрии:

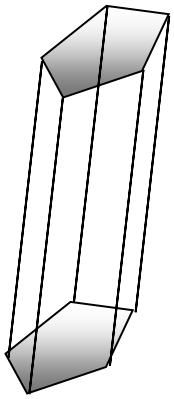
Додекаэдр имеет центр симметрии - центр додекаэдра, 15 осей симметрии и 15 плоскостей симметрии.

Икосаэдр

Икосаэдр составлен из двадцати равносторонних треугольников. Каждая его вершина является вершиной пяти треугольников. Сумма плоских углов при каждой вершине равна 300 градусов. Таким образом, икосаэдр имеет 20 граней, 12 вершин и 30 ребер.

Элементы симметрии:

Икосаэдр имеет центр симметрии - центр икосаэдра, 15 осей симметрии и 15 плоскостей симметрии.



Пятиугольная призма (карандаш)

Пятиугольная призма составлена из двух равносторонних пятиугольников в основании, соединенных прямыми линиями.

Элементы симметрии:

Пятиугольная призма имеет две оси симметрии.

Призматоид

Призматоид — многогранник, две грани которого (основания призматоида) лежат в параллельных плоскостях, а остальные являются треугольниками (или трапециями), причём у треугольников одна сторона, а у трапеций оба основания являются сторонами оснований призматоида.

Приложение В

Введение в OpenGL

Цель работы – изучение принципов применения библиотеки OpenGL при разработке приложений в C#.

OpenGL

OpenGL означает *Open Graphics Library*, что переводится как «открытая графическая библиотека».

Другими словами, *OpenGL* − это некая спецификация, включающая в себя несколько сотен функций. Она определяет независимый от языка программирования кросс-платформенный программный интерфейс, с помощью которого программист может создавать приложения, использующие двухмерную и трехмерную компьютерную графику. Первая базовая версия *OpenGL* появилась в 1992 году, она была разработана компанией *Silicon Graphics Inc*, занимающейся разработками в области трехмерной компьютерной графики.

В библиотеку заложен механизм расширений, благодаря которому производители аппаратного обеспечения (например, производители видеокарт) могли выпускать расширения *OpenGL* для поддержки новых специфических возможностей, не включенных в текущую версию библиотеки. Благодаря этому программисты могли сразу использовать эти новые возможности в отличие от библиотеки *Microsoft Direct3D* (в этом случае им бы пришлось ждать выхода новой версии *DirectX*).

Библиотеки *OpenGL* и *DirectX* являются конкурентами на платформе *MS Windows*. *Microsoft* продвигает свою библиотеку *DirectX* и стремится замедлить развитие библиотеки *OpenGL*, что ослабило бы графическую систему конкурирующих ОС, где используется исключительно библиотека *OpenGL* для реализации вывода всей графики.

Мы будем учиться визуализации компьютерной графики именно с применением этой библиотеки. Однако прямой поддержки данной библиотеки в *.NET Framework* нет, поэтому мы будем использовать библиотеку *Tao Framework*.

TAO Framework

Tao Framework – это свободно распространяемая библиотека с открытым исходным кодом, предназначенная для быстрой и удобной разработки кросс-платформенного мультимедийного программного обеспечения в среде *.NET Framework* и *Mono*.

На сегодняшний день *Tao Framework* − оптимальный путь для использования библиотеки *OpenGL* при разработке приложений в среде *.NET* на языке *C#*.

В состав библиотеки на данный момент входят все современные средства, которые могут понадобиться в ходе разработки мультимедиа программного обеспечения: реализация библиотеки *OpenGL*, реализация библиотеки *FreeGlut*, содержащей все самые новые функции этой библиотеки, библиотека *DevIL* (легшая в основу стандарта *OpenIL − Open Image Library*) и многие другие.

Самые интересные библиотеки, включенные в *Tao Framework*:

- *OpenGL 2.1.0.12* − свободно распространяемый аппаратнопрограммный интерфейс для визуализации 2D- и 3D-графики.
- *FreeGLUT 2.4.0.2* – библиотека с открытым исходным кодом, являющаяся альтернативой библиотеке *GLUT* (*OpenGL Utility Toolkit*).
- *DevIL 1.6.8.3* (она же *OpenIL*) – кросс-платформенная библиотека, реализующая программный интерфейс для работы с изображениями. На данный момент библиотека поддерживает работу с изображениями 43 форматов для чтения и 17 форматов для записи.
- *Cg 2.0.0.0* – язык высокого уровня, созданный для программирования текстурных и вершинных шейдеров.
- *OpenAL 1.1.0.1* – свободно распространяемый аппаратно-программный интерфейс для обработки аудиоданных. (В том числе 3D-звука и EAX эффектов).
- *PhysFS 1.0.1.2* – библиотека для работы с вводом/выводом файловой системы, а также различного вида архивами на основе собственного *API*.
- *SDL 1.2.13.0* – кросс-платформенная мультимедийная библиотека, активно использующаяся для написания мультимедийных приложений в операционной системе *GNU/Linux*.

• *ODE 0.9.0.0* − свободно распространяемый физический программный интерфейс, главной особенностью которого является реализация системы динамики абсолютно твёрдого тела и системы обнаружения столкновений.

• *FreeType 2.3.5.0* – библиотека, реализующая растеризацию шрифтов. Данная библиотека используется в *X11* − оконной системе, которая обеспечивает все стандартные инструменты и протоколы для построения *GUI* (графического интерфейса пользователя) в *UNIX* подобных операционных системах.

• *FFmpeg 0.4.9.0* − набор свободно распространяемых библиотек с открытым исходным кодом. Данные мультимедийные библиотеки позволяют работать с аудио- и видеоданными в различных форматах.

Таким образом, библиотека *Tao Framework* является мощным и удобным свободно распространяемым инструментом для решения любых мультимедийных задач, преимущественно кросс-платформенного характера. Работая с данной библиотекой, разработчики могут использовать базу алгоритмов и реализованных за многие годы методов, что сокращает время разработки программных продуктов.

Создание проекта и подключение библиотеки *Tao OpenGL* в *C#*

Сначала создайте новый проект, в качестве шаблона установив приложение Windows Forms. Назовите его, например, OpenGLTest.

Дождитесь, пока MS Visual Studio закончит генерацию кода шаблона.

Назовите главное окно «Примитивы OpenGL».

Теперь перейдите к окну Solution Explorer (Обозреватель решений). Здесь нас интересует узел References (Ссылки), который отображает связи с библиотеками, необходимыми для работы нашего приложения (рисунок В.1).

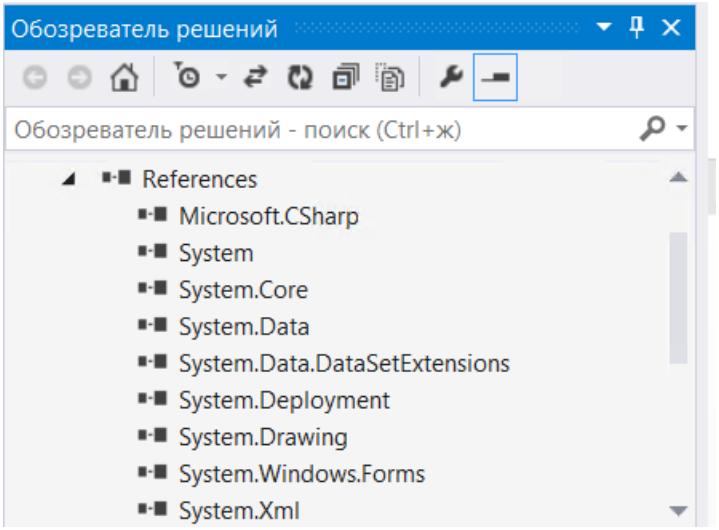
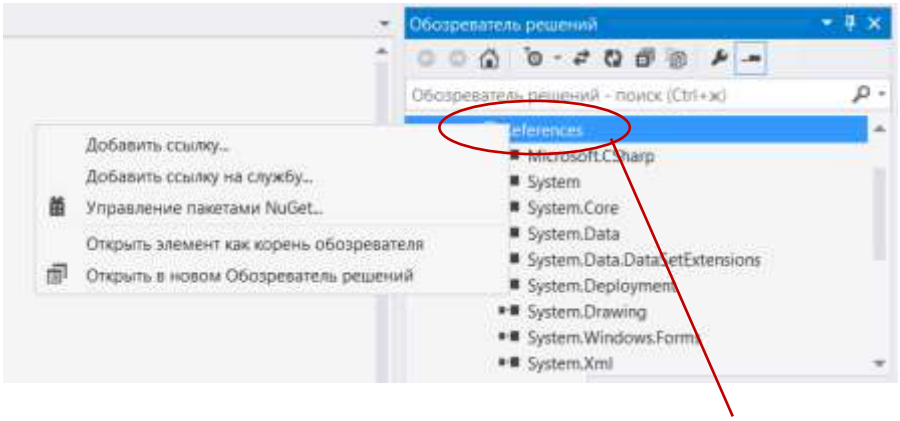


Рисунок В.1

Щелкните по этому узлу правой клавишей мыши, после чего в открывшемся контекстном меню выберите «Добавить ссылку» (“Add Link”), как показано на рисунок В.2.



Щелчок правой кнопкой мыши

Рисунок В.2

В открывшемся окне «Добавить ссылку» перейдите к закладке «Обзор» и нажмите кнопку «Обзор». После этого перейдите к директории, в которую была установлена библиотека *Tao Framework*. (В нашем случае это папка “*C:\Program Files (x86)\Microsoft Visual Studio 12.0\OpenGL*”).

Нам потребуется папка *bin*, в ней хранятся необходимые нам библиотеки.

Перейдите в папку *bin* и выберите три библиотеки: ***Tao.OpenGL.dll, Tao.FreeGlut.dll, Tao.Platform.Windows.dll***

Tao.OpenGL.dll отвечает за реализацию библиотеки *OpenGL*.

Tao.FreeGlut.dll отвечает за реализацию функций библиотеки *Glut*. Мы будем ее использовать для инициализации рендера, а также для других целей.

Tao.Platform.Windows.dll отвечает за поддержку элементов для визуализации на платформе *Windows*.

На рисунке В.3 мы видим все добавившиеся библиотеки в узле «*References*» (Ссылки).

Теперь перейдите к исходному коду окна. Для работы с нашими библиотеками необходимо подключить соответствующие пространства имен:

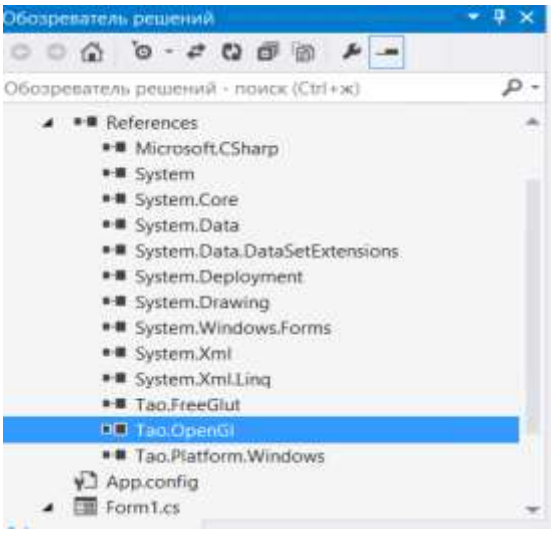


Рисунок В.3

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Threading;
// для работы с библиотекой OpenGL
using Tao.OpenGl;
// для работы с библиотекой FreeGLUT
using Tao.FreeGlut;
// для работы с элементом управления SimpleOpenGLControl
using Tao.Platform.Windows;
```

Теперь вернитесь к конструктору диалогового окна и перейдите к Панели элементов. Щелкните правой кнопкой на вкладке «Общие» и в раскрывшемся контекстном меню выберите пункт «Выбрать элементы» (Choose Items), как показано на рисунке В.4.

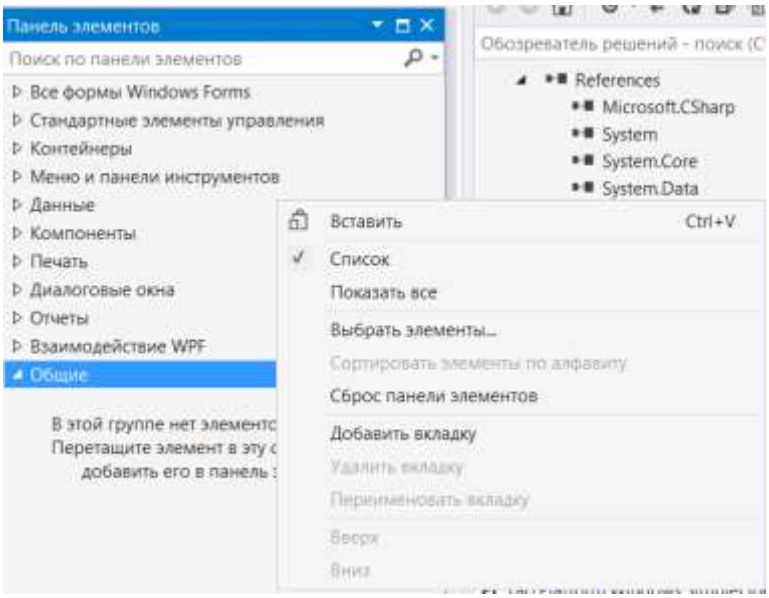


Рисунок В.4

В открывшемся окне найдите элемент *SimpleOpenGLControl* и установите возле него галочку, как показано на рисунке 7.5. Если в представленном списке элементов нет элемента *SimpleOpenGLControl*, нажмите кнопку «Обзор» и в появившемся окне выберите и откройте библиотеку *Tao.Platform.Windows.dll*. Затем выберите галочкой *SimpleOpenGLControl* и нажмите ОК.

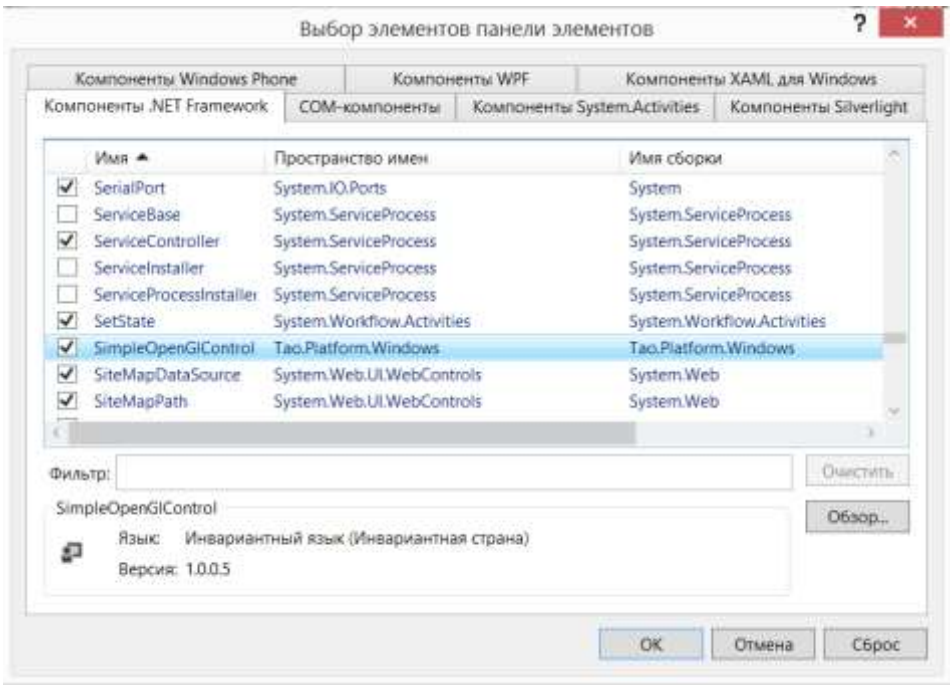


Рисунок В.5

Теперь данный элемент станет доступным для размещения на форме приложения. Перетащите элемент на форму и разместите так, как показано на рисунке В.6. Справа от размещенного элемента установите необходимые элементы.

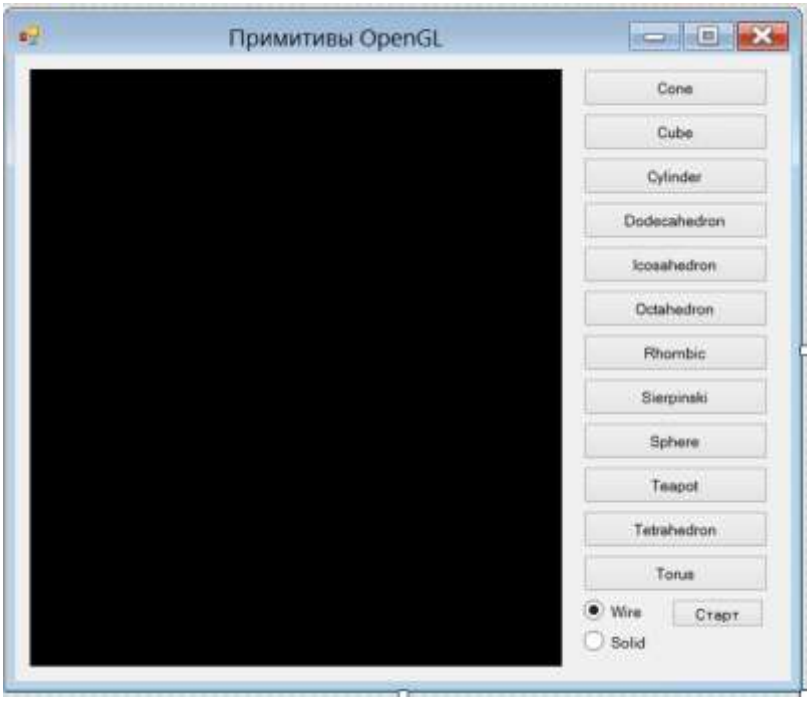


Рисунок В.6

Выделите элемент *simpleOpenGlControl1*, расположенный на форме и перейдите к его свойствам. Измените параметр *name* на значение «*Pr*». Далее элементы *simpleOpenGlControl* будем называть *Pr*.

Инициализация *OpenGL* и *Glut* в *C#*

Теперь необходимо инициализировать работу *OpenGl*. Сначала в конструкторе класса необходимо инициализировать работу элемента *Pr*:

```
public Form1()
{
    InitializeComponent();
    // Инициализация контекста окна графического вывода
    Pr.InitializeContexts();
}
```

Снова перейдите к конструктору и сделайте двойной щелчок левой клавишей мыши на форме – создастся функция обработчик события загрузки формы.

В ней поместим код инициализации *OpenGl и Glut*. Подробное описание кода визуализации объекта рассмотрено в приложении Г.

```
        private void Form1_Load(object sender, EventArgs e)
        {
            // Черный цвет фона
            Gl.glClearColor(0, 0, 0, 1);
            // Инициализация Glut
            Glut.glutInit();
            // Используем в Glut систему цветов RGB, двойную буферизацию
(экранный и внеэкранный буферы) и буфер глубины
            Glut.glutInitDisplayMode(Glut.GLUT_RGB | Glut.GLUT_DOUBLE |
Glut.GLUT_DEPTH);
            // Активизируем тест глубины
            Gl.glEnable(Gl.GL_DEPTH_TEST);
        }
```

Откомпилируйте и запустите приложение.

Если при компиляции приложения *Microsoft Visual Studio* выдает ошибку с сообщением о том, что не найдена какая-либо библиотека *OpenGL* (например, *freeglut.dll*), необходимо в настройках операционной системы в переменную среды *PATH* добавить каталог данной библиотеки. При невозможности это сделать, допустимо из каталога *C:\Program Files (x86)\Microsoft Visual Studio 12.0\OpenGL\lib* скопировать все файлы необходимых библиотек (в нашем случае – *freeglut.dll*) в каталог *Debug* нашего проекта.

Реализация программы вывода примитивов библиотеки Glut

Реализуем программный код вывода стандартных примитивов:

- конус;
- куб;
- цилиндр;
- додекаэдр;
- икосаэдр;
- октаэдр;
- ромбический додекаэдр;
- губка Серпиского;
- сфера;

- чайник;
- тетраэдр;
- тор.

Объекты представлены в библиотеке *FreeGlut* как полигональные модели. Их можно отобразить в виде каркаса (*glutWire*) либо залитыми цветом (*glutSolid*) или текстурой (*glutSolid* + текстура, в данном случае необходимо еще подключить библиотеку *DevIL.dll*).

Программа обеспечивает вывод любого из перечисленных выше представлений.

При проецировании используется изометрия, получаемая в результате умножения единичной матрицы на матрицы поворота по часовой стрелки на 30° и против нее на 45° соответственно вокруг осей X и Y:

```
// Матрица проецирования
Gl.glMatrixMode(Gl.GL_PROJECTION);
Gl.glLoadIdentity();
Gl.glRotated(-30, 1, 0, 0);
Gl.glRotated(45, 0, 1, 0);
```

Видовая матрица преимущественно является единичной:

```
// Видовая матрица
Gl.glMatrixMode(Gl.GL_MODELVIEW);
Gl.glLoadIdentity();
```

При выводе додекаэдра единичная видовая матрица умножается на матрицу масштабирования:

```
Gl.glScaled(0.5, 0.5, 0.5);
```

При выводе тонированных тел (*glutSolid*) используется модель освещенности с одним источником света. Нормали к glut-объектам генерируются автоматически.

Заметим, что Платоновы тела выводятся без сглаживания, даже если указан

```
Gl.glShadeModel(Gl.GL_SMOOTH); // Вывод с интерполяцией цветов
```

Примитив выводится после нажатия на одну из кнопок.

В зависимости от положения переключателя (*Wire* или *Solid*) отображается либо каркасная, либо тоновая, либо текстурированная модель (рисунок В.7).

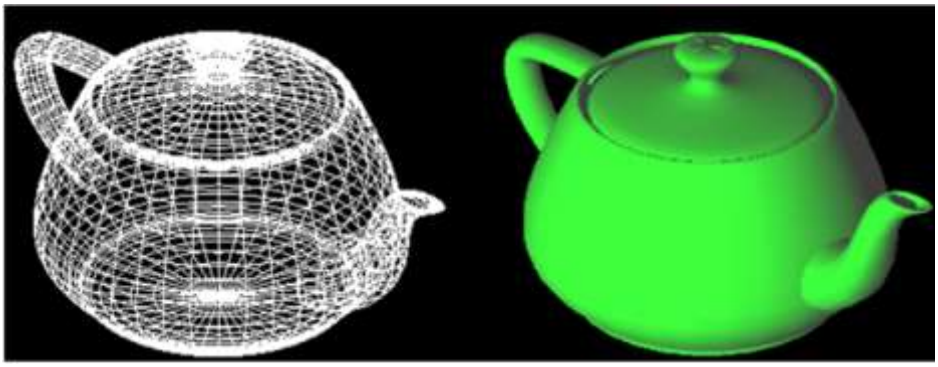


Рисунок В.7 - Два способа вывода примитива

Полный код приложения:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Threading;

using Tao.OpenGl;
using Tao.FreeGlut;
using Tao.Platform.Windows;


namespace OGL_Primet
{
    public partial class Form1 : Form
    {

        public Form1()
        {
            InitializeComponent();
            // Инициализация контекста окна графического вывода
            Pr.InitializeContexts();
        }


        private void showSolid(int obj)
        {
            switch (obj)
            {
                case 1:
                    Glut.glutSolidCone(0.2, 0.75, 16, 8); break; // Конус
                case 2:
                    Glut.glutSolidCube(0.75); break; // Куб
                case 3:
                    Glut.glutSolidCylinder(0.2, 0.75, 16, 16); break;//Цилиндр
```

```
                case 4:
                    Gl.glScaled(0.5, 0.5, 0.5);
                    Glut.glutSolidDodecahedron(); break; // Додекаэдр
                case 5:
                    Glut.glutSolidIcosahedron(); break; // Икосаэдр
                case 6:
                    Glut.glutSolidOctahedron(); break; // Октаэдр
                case 7:
                    Glut.glutSolidRhombicDodecahedron(); break; // Ромбический
додекаэдр
                case 8:
                    double[] offset = { 0.0 };
                    Glut.glutSolidSierpinskiSponge(7, offset, 1); break; // Фрактал
Губка Серпиского
                case 9:
                    Glut.glutSolidSphere(0.75, 16, 16); break; // Сфера
                case 10:
                    Glut.glutSolidTeapot(0.5); break; // Чайник
                case 11:
                    Gl.glRotated(180, 0, 1, 0);
                    Glut.glutSolidTetrahedron(); break; // Тетраэдр
                case 12:
                 Glut.glutSolidTorus(0.15, 0.65, 16, 16); break; // Тор
            }
        }


        private void draw(int obj)
        {
            // Очистка буфера цвета и буфера глубины
            Gl.glClear(Gl.GL_COLOR_BUFFER_BIT | Gl.GL_DEPTH_BUFFER_BIT |
Gl.GL_ACCUM_BUFFER_BIT);
            // Матрица проецирования
            Gl.glMatrixMode(Gl.GL_PROJECTION);
            Gl.glLoadIdentity();
            Gl.glRotated(-30, 1, 0, 0);
            Gl.glRotated(45, 0, 1, 0);
            // Видовая матрица
            Gl.glMatrixMode(Gl.GL_MODELVIEW);
            Gl.glLoadIdentity();
            if (radioButton1.Checked)
            {
                // Белый цвет
                Gl.glColor3f(1, 1, 1);
                // Выводим glut-примитив в виде каркаса
                switch (obj)
                {
                    case 1:
                        Glut.glutWireCone(0.2, 0.75, 16, 8); break; // Конус
                    case 2:
                        Glut.glutWireCube(0.75); break; // Куб
                    case 3:
                        Glut.glutWireCylinder(0.2, 0.75, 16, 16); break; // Цилиндр
                    case 4:
                        Gl.glScaled(0.5, 0.5, 0.5);
                        Glut.glutWireDodecahedron(); break; // Додекаэдр
                    case 5:
                        Glut.glutWireIcosahedron(); break; // Икосаэдр
                    case 6:
                        Glut.glutWireOctahedron(); break; // Октаэдр
                    case 7:
```

```
                    Glut.glutWireRhombicDodecahedron(); break; // Ромбический
додекаэдр
                case 8:
                    double[] offset = { 0, 0, 0 };
                    Glut.glutWireSierpinskiSponge(7, offset, 1); break; // Фрактал
Губка Серпиского
                case 9:
                    Glut.glutWireSphere(0.75, 16, 16); break; // Сфера
                case 10:
                    Glut.glutWireTeapot(0.5); break; // Чайник
                case 11:
                    Gl.glRotated(180, 0, 1, 0);
                    Glut.glutWireTetrahedron(); break; // Тетраэдр
                case 12:
                    Glut.glutWireTorus(0.15, 0.65, 16, 16); break; // Тор
            }
        }
        else if (radioButton2.Checked)
        {
            // Модель освещенности с одним источником цвета
            float[] light_position = { 10, 10, -30, 0 }; // Координаты источника
света
            float[] lghtClr = { 1, 1, 1, 0 }; // Источник излучает белый цвет
            float[] mtClr = { 0, 1, 0, 0 }; // Материал зеленого цвета
            Gl.glPolygonMode(Gl.GL_FRONT, Gl.GL_FILL); // Заливка полигонов
            Gl.glShadeModel(Gl.GL_SMOOTH); // Вывод с интерполяцией цветов
            Gl.glEnable(Gl.GL_LIGHTING); // Будем рассчитывать освещенность
            Gl.glLightfv(Gl.GL_LIGHT0, Gl.GL_POSITION, light_position);
            Gl.glLightfv(Gl.GL_LIGHT0, Gl.GL_AMBIENT, lghtClr); // Рассеивание
            Gl.glEnable(Gl.GL_LIGHT0); // Включаем в уравнение освещенности
источник GL_LIGHT0
            // Диффузионная компонента цвета материала
            Gl.glMaterialfv(Gl.GL_FRONT, Gl.GL_DIFFUSE, mtClr);
            // Выводим тонированный glut-примитив
            showSolid(obj);
            Gl.glDisable(Gl.GL_LIGHTING); // Будем рассчитывать освещенность
        }
        Pr.Invalidate();
    }


    private void button1_Click(object sender, EventArgs e)
    {
        draw(1); // Конус
    }

    private void button2_Click(object sender, EventArgs e)
    {
        draw(2); // Куб
    }

    private void button3_Click(object sender, EventArgs e)
    {
        draw(3); // Цилиндр
    }

    private void button4_Click(object sender, EventArgs e)
    {
        draw(4); // Додекаэдр
    }

    private void button5_Click(object sender, EventArgs e)
```

```
        {
            draw(5); // Икосаэдр
        }

        private void button6_Click(object sender, EventArgs e)
        {
            draw(6); // Октаэдр
        }

        private void button7_Click(object sender, EventArgs e)
        {
            draw(7); // Ромбический додекаэдр
        }

        private void button8_Click(object sender, EventArgs e)
        {
            draw(8); // Фрактал Губка Серпиского
        }

        private void button9_Click(object sender, EventArgs e)
        {
            draw(9); // Сфера
        }

        private void button10_Click(object sender, EventArgs e)
        {
            draw(10); // Чайник
        }

        private void button11_Click(object sender, EventArgs e)
        {
            draw(11); // Тетраэдр
        }

        private void button12_Click(object sender, EventArgs e)
        {
            draw(12); // Тор
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // Черный цвет фона
            Gl.glClearColor(0, 0, 0, 1);
            // Инициализация Glut
            Glut.glutInit();
            // Используем в Glut систему цветов RGB, двойную буферизацию (экранный и
внеэкранный буферы) и буфер глубины
            Glut.glutInitDisplayMode(Glut.GLUT_RGB | Glut.GLUT_DOUBLE |
Glut.GLUT_DEPTH);
            // Активизируем тест глубины
            Gl.glEnable(Gl.GL_DEPTH_TEST);
        }
    }
}
```