**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»**

*КАФЕДРА ИНЖЕНЕРНОЙ КИБЕРНЕТИКИ*

*НАПРАВЛЕНИЕ ПРИКЛАДНАЯ МАТЕМАТИКА ГРУППА БПМ-16-1*

**Домашнее задание**

*ПО КУРСУ*: «Математические методы в компьютерной графике»

*ТЕМА*: «Графическая система OpenGL. Буфер кадра.»

*СТУДЕНТ: Ухова Александра*

*ПРЕПОДАВАТЕЛЬ: Куренкова Т. В.*

Москва, 2020 г.

Оглавление

[Введение 3](#_Toc34823440)

[Аналитический обзор 5](#_Toc34823441)

[Буферы и их применение 5](#_Toc34823442)

[Буферы цвета 5](#_Toc34823443)

[Буфер глубины 6](#_Toc34823444)

[Буфер трафарета 6](#_Toc34823445)

[Буфер накопления 6](#_Toc34823446)

[Очистка буферов 7](#_Toc34823447)

[Тесты и операции над фрагментами 8](#_Toc34823448)

[Тест на отсечение 9](#_Toc34823449)

[Альфа-тест 9](#_Toc34823450)

[Тест трафарета 10](#_Toc34823451)

[Тест глубины 12](#_Toc34823452)

[Смешивание, размытие и логические операции 12](#_Toc34823453)

[Смешивание 13](#_Toc34823454)

[Размытие 13](#_Toc34823455)

[Логические операции 14](#_Toc34823456)

[Буфер накопления 15](#_Toc34823457)

[Сглаживание сцены 16](#_Toc34823458)

[Глубина резкости 17](#_Toc34823459)

[Содержательная постановка задачи 18](#_Toc34823460)

[Демонстрация работы программы 19](#_Toc34823461)

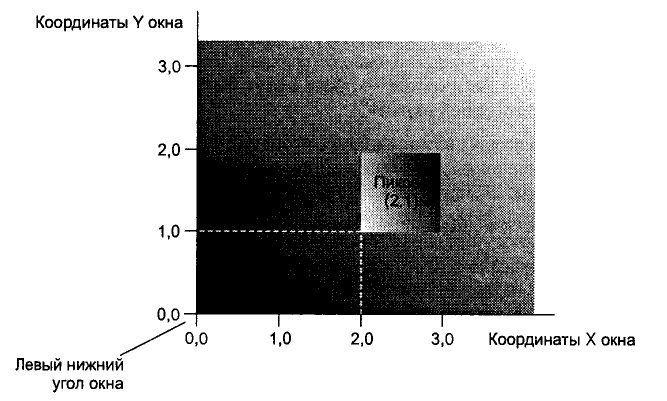
[Выводы 23](#_Toc34823462)

[Список используемой литературы 24](#_Toc34823463)

# Введение

Конечной целью практически любой графической программы является вывод изображения на экран. Экран представляет собой прямоугольный массив пикселов, каждый из которых отображает очень маленький окрашенный квадрат, соответствующий точке изображения. После этапа растеризации (включающего в себя текстурирование и наложение тумана) данные ещё не являются пикселами, пока они представляют собой фрагменты. Каждый фрагмент описывается координатной информацией, а также значениями цвета и глубины. Каждый такой фрагмент подвергается серии тестов и операций. Прошедшие тесты фрагменты готовы к тому, чтобы стать пикселами. Чтобы нарисовать эти пикселы, необходимо знать их цвет, то есть информацию, сохранённую в буфере цвета. Хранилище, в которое единообразно помещаются все пикселы, называется буфером. Различные буферы могут содержать различное количество информации о пикселе, но внутри каждого буфера пиксел описывается одной и той же информацией. Буфер, сохраняющий один бит информации о пикселе, называется битовой плоскостью (bitplane).

Как показано на рис. 1, левый нижний пиксел окна OpenGL, (0, 0), в оконных координатах соответствует левому нижнему углу области размерами 1 х 1, занимаемой пикселом. В общем случае пиксел (x, y) заполняет собой область, ограниченную значением x слева, значением x + 1 справа, y – снизу и y + 1 – сверху.



*Рис. 1. Область, занимаемая пикселом.*

Рассмотрим подробнее буфер, который хранит цветовую информацию о данных, отображаемых на экране. Положим, что ширина экрана составляет 1280 пикселов, а высота – 1024 пиксела, и что мы имеем дело с 24-разрядным экраном – таким, который может воспроизвести различных цветов. Поскольку 24 бита – это то же, что и 3 байта, буфер цвета должен быть разбит на порции из 3 байт для каждого из 1 310 720 (произведение 1280 на 1024) пикселов экрана. Число пикселов на физическом экране зависит от возможностей аппаратного обеспечения, но количество информации о пикселе, сохраняемой в буфере цвета, – величина одинаковая во всех случаях.

Буфер цвета – это только один из нескольких буферов, хранящих информацию о пикселах. Так, к примеру, буфер глубины содержит информацию о глубине пикселов. Сам буфер цвета может состоять из нескольких подбуферов. Понятие буфера кадров (framebuffer) относится ко всем этим буферам. За исключением буфера цвета, невозможно увидеть другие буферы непосредственно; они привлекаются для таких задач, как удаление невидимых поверхностей, сглаживание сцены, отображение по маске, воспроизведение плавного движения и других операций.

Данная работа разбита на следующие основные разделы:

* Буферы и их применение. Рассматриваются существующие буферы, их назначение и управление ими – очистку и разрешение записи.
* Тесты и операции над фрагментами. Рассматривается сущность тестов на отсечение трафарета, глубины и альфа-теста, выполняемых после расчёта положения и цвета пиксела, но до его вывода на экран. Несколько операций – смешивание, размытие и логические – также могут быть выполнены перед тем, как фрагмент обновит свой экран.
* Буфер накопления. Описывает более сложные операции, включающие в себя сглаживание сцены, размытие при движении и имитацию глубины, подобно фотографиям.

# Аналитический обзор

## Буферы и их применение

Система OpenGL может управлять следующими буферами:

* цветовыми: передним левым, передним правым, задним левым, задним правым и любым количеством вспомогательных буферов;
* буфером глубины;
* буфером трафарета;
* буфером накопления.

Конкретная реализация OpenGL определяет, какие буферы доступны и какое количество битов на пиксел для них принято. Кроме этого, поддерживаются составные, то есть «оконные» типы, подразумевающие работу с несколькими буферами. Таблица 1 содержит список параметров функции glGetIntegerv(), которая запрашивает систему OpenGL о размере памяти, отводимой пикселу какого-либо объекта визуализации.

|  |  |
| --- | --- |
| Параметр | Пояснение |
| GL\_RED\_BITS, GL\_GREEN\_BITS, GL\_BLUE\_BITS, GL\_ALPHA\_BITS | Количество битов на R-, G-, B- и A-компоненты цветовых буферов |
| GL\_INDEX\_BITS | Количество битов на индекс в буферах цвета |
| GL\_DEPTH\_BITS | Количество битов на пиксел в буфере глубины |
| GL\_STENCIL\_BITS | Количество битов на пиксел в буфере трафарета |
| GL\_ACCUM\_RED\_BITS, GL\_ACCUM\_GREEN\_BITS, GL\_ACCUM\_BLUE\_BITS, GL\_ACCUM\_ALPHA\_BITS | Количество битов на R-, G-, B- и A-компоненты буфера накопления |

*Таблица 1. Параметры запроса о размере памяти, отводимой под пиксел.*

### Буферы цвета

Буфера цвета – это то место, где обычно выполняется рисование. Они содержат RGB- или индексные данные и могут также содержать значения альфа-компонента. В реализациях OpenGL, поддерживающих стереоскопический просмотр, имеются левый и правый буферы цвета для стереоизображений. В противном случае используются только левые буферы. Аналогично, система двойной буферизации подразумевает рабочий и фоновый буферы, а система однократной буферизации работает только с рабочими буферами. Любая реализация OpenGL предоставляет передний левый буфер цвета.

Необязательно, но могут поддерживаться неотображаемые буферы. OpenGL не регламентирует правила их применения, поэтому их можно использовать по своему усмотрению. Например, задействовать их для хранения многократно используемых изображений. Тогда, чтобы не перерисовывать изображение заново, можно скопировать существующее из вспомогательного буфера в нужные буфера цвета.

Также существуют параметры GL\_STEREO и GL\_DOUBLEBUFFER для команды glGetBooleanv(), позволяющие выяснить, поддерживается ли в системе стерео-вывод (то есть доступен ли правый буфер) или двойная буферизация (задний буфер). Чтобы узнать, какие вообще вспомогательные буферы присутствуют, используйте команду glGetIntegerv() с параметром GL\_AUX\_BUFFERS.

### Буфер глубины

Буфер глубины хранит значение глубины для каждого пиксела. Под глубиной понимается расстояние до наблюдателя исходя из того, что пикселы с большей глубиной записываются поверх пикселов с меньшими значениями. Это полезное соглашение, хотя поведение буфера глубины может быть изменено. Буфер глубины иногда также называют z-буфером (z – это как бы третья координата (помимо x и y), перпендикулярная экрану).

### Буфер трафарета

Одно из применений буфера трафарета – это ограничение области рисования какой-то частью экрана, аналогично тому, как шаблон из картона помогает создать резкие изображения при простом распылении красителя. Например, можно воспроизвести картину, видимую через лобовое стекло, имеющее неправильную геометрическую форму. Тогда нужно будет сохранить форму ветрового стекла в буфере трафарета и в дальнейшем использовать его для вывода полной сцены. Буфер трафарета предотвратит отображение лишней панорамы, невидимой через переднее стекло. Таким образом, если приложение моделирует автомобиль, достаточно один раз нарисовать приборную доску и другие элементы внутри автомобиля, а при имитации его движения только обновлять внешнюю среду.

### Буфер накопления

Буфер накопления хранит данные цвета RGBA, как и каждый буфер цвета в режиме RGBA. (В режиме индексированного цвета для буфера накопления результаты не определены.) Обычно накапливается серия изображений, затем, сливаемое в конечное, составное изображение. Таким образом, можно выполнять такие операции, как сглаживание, с помощью избыточной выборки (supersampling) и усреднения отсчётов, с целью последующей окраски пикселов в буфере цвета в полученное значение цвета. Рисование в буфер накопления непосредственно не производится; он просто аккумулирует действия, всегда выполняемые прямоугольными областями, поступающими в буфер цвета и исходящими из него.

### Очистка буферов

В графических программах очистка экрана (или любого из буферов) – обычно одна из самых «дорогостоящих» операций. Так, на мониторе 1280\*1024 при очистке экрана требуется обработать более миллиона пикселов. В простых графических приложениях операция очистки может отнять больше времени, чем всё прочее рисование. Если необходимо очистить не только буфер цвета, но также буферы глубины и трафарета, время очистки увеличится ещё в три раза.

Чтобы решить эту проблему, некоторое аппаратное обеспечение очищает все буферы вместе. Команды OpenGL для очистки по возможности оптимизированы под такую архитектуру. Вначале необходимо указать значения, которые следует записать в каждый буфер для его сброса. Затем даётся единственная команда на выполнение очистки, самостоятельно перебирающая буферы из данного списка. Внутри же всё зависит от архитектуры, если она способна на единовременную очистку, таковая и выполняется, в противном случае буферы очищаются последовательно.

Следующие команды устанавливают значения очистки для каждого буфера:

* void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);
* void glClearIndex(GLfloat index);
* void glClearDepth(GLclampf depth);
* void glClearStencil(GLint s);
* void glClearAccum(GLfloat red, GLfloat, green, GLfloat, GLfloat alpha)

Эти команды задают значения очистки для буферов цвета (в режиме RGBA или индексации цвета), глубины, трафарета или накопления соответственно. Типы GLclampf и GLclampd (ограниченные GLfloat и GLdouble соответственно) имеют диапазон значений от 0 до 1. Значение по умолчанию для глубины – 1.0; все другие значения по умолчанию равны 0. Однажды установленные значения имеют силу до тех пор, пока они не будут изменены следующим вызовом той же команды.

Непосредственно очистка буфера производится вызовом glClear():

* void glClear(GLbitfield mask).

Маска (mask) – это поразрядное логическое ИЛИ из констант GL\_COLOR\_BUFFER\_BIT, GL\_DEPTH\_BUFFER\_BIT, GL\_STENCIL\_BUFFER\_BIT и GL\_ACCUM\_BUFFER\_BIT, идентифицирующих собой конкретный тип буфера. Очистка с параметром GL\_COLOR\_BUFFER\_BIT выполняется согласно текущему режиму цвета – RGBA или индексному. При этом очищаются все буферы, доступные для записи. Если включены тексты принадлежности (ownership), отсечения и размытия, они применяются к операциям очистки. Также здесь могут быть применены маскирующие операции, такие как glColorMask() и glIndexMask(). Альфа-тест, тесты трафарета и глубины никак не воздействуют на glClear().

## Тесты и операции над фрагментами

Когда происходит отображение геометрических фигур, текстов или изображений, OpenGL выполняет определённые расчёты для поворота, переноса, масштабирования, определения освещения, получения проекции объекта в перспективе, вычисляет, какие пикселы следует воспроизводить в окне, и выясняет, какими цветами должны быть нарисованы эти пикселы. Затем OpenGL решает, какие фрагменты следует генерировать и с какими цветами. Часть этапов оставшейся обработки зависит от местонахождения соответствующего фрагменту пиксела в буфере кадров. Например, если он расположен снаружи прямоугольной области или если находится дальше от точки наблюдения, чем пиксел, уже помещённый в буфер кадров, фрагмент не рисуется. На других стадиях цвета фрагментов смешиваются с цветом пиксела, присутствующего в буфере кадров.

В этом разделе описывается два полных набора тестов: выполняемых до поступления фрагмента в буфер кадров и заключительных действий. Тесты и операции выполняются в следующем порядке (если фрагмент исключается из рассмотрения ранними тестами, к нему не применяются дальнейшие тесты и операции):

1. Тест на отсечение
2. Альфа-тест
3. Тест трафарета
4. Тест глубины
5. Смешивание
6. Размытие
7. Логические операции

Все перечисленные тесты и операции подробно рассматриваются в следующих подразделах.

### Тест на отсечение

Определить прямоугольную область окна и ограничить рисование можно только её пределами, используя команду glScissor(). Если фрагмент лежит внутри прямоугольника, он считается успешно прошедшим тест на отсечение.

void glScissor(Glint x, Glint y, GLsizei width, GLsizei heigh).

Данная функция устанавливает положение и размеры прямоугольника отсечения (scissor box). Эти параметры определяют левый нижний угол (x, y), ширину и высоту прямоугольника. Пикселы, лежащие внутри прямоугольника, успешно проходят тест. Отсечение включается и выключается указанием константы GL\_SCISSOR\_TEST в вызовах glEnable() и glDisable(). По умолчанию размеры прямоугольника приравниваются размерам окна и отсечение выключено.

Тест на отсечение – это частный случай теста трафарета, где в качестве области отсечения используется прямоугольная область. Выделение его в особый случай объясняется тем, что на аппаратном уровне прямоугольное отсечение реализуется намного проще. Если же отсечение происходит медленно, значит оно реализовано чере з вывод по шаблону программно.

В расширенном варианте отсечение выполняется посредством нелинейного проецирования. Сначала окно делится регулярной сеткой на подобласти, указываются область просмотра и параметры отсечения, которые ограничивают единовременное воспроизведение областей. Далее полная сцена проецируется на каждый регион с использованием его собственной матрицы проекции.

Чтобы узнать, активно ли отсечение и запросить параметры прямоугольника отсечения, необходимо использовать константу GL\_SCISSOR\_TEST для glIsEnabled() и GL\_SCISSOR\_BOX для glGetIntegerv().

### Альфа-тест

В режиме RGBA альфа-тест позволяет вам принять или отклонить фрагмент, основываясь на значении альфа-компонента. Альфа-тест включается и выключается заданием константы GL\_ALPHA\_TEST для вызовов glEnable() и glDisable(). Чтобы определить, активен ли этот тест, необходимо использовать вызов glIsEnabled() с параметром GL\_ALPHA\_TEST.

При работе теста сравнивается входящее значение альфа-компонента с его образцовым значением. Успех прохождения фрагмента через тест зависит от результата сравнения. Как эталонная величина, так и функция сравнения устанавливаются командой glAlphaFunc(). По умолчанию в качестве образца принимается 0, функция сравнения приравнивается GL\_ALWAYS, а альфа-тестирование не действует. Чтобы получить информацию о параметрах тестирования, используйте константы GL\_ALPHA\_TEST\_FUNC и GL\_ALPHA\_TEST\_REF при вызове glGetIntegerv().

void glAlphaFunc(GLenum func, GLclampf ref)

Данная функция устанавливает эталонное значение и функцию сравнения для альфа-теста. Эталонное значение (ref) ограничивается интервалом от 0 до 1. Допустимые значения аргумента func и их толкование представлены в табл. 2. Обозначению соответствует фрагменту, а обозначение – эталонному значению.

|  |  |
| --- | --- |
| Параметр | Пояснение |
| GL\_NEVER | Никогда не принимать фрагменты |
| GL\_ALWAYS | Всегда принимать фрагменты |
| GL\_LESS | Принять фрагмент, если |
| GL\_LEQUAL | Принять фрагмент, если |
| GL\_EQUAL | Принять фрагмент, если |
| GL\_GEQUAL | Принять фрагмент, если |
| GL\_GREATER | Принять фрагмент, если |
| GL\_NOTEQUAL | Принять фрагмент, если |

*Таблица 2. Значения параметров glAlphaFunc().*

Одно из применений альфа-теста – реализация прозрачности. Для этого сцена воспроизводится дважды: сначала с фрагментами, у которых значение альфа-компонента равно 1, а затем с фрагментами, где это значение не равно единице. На обоих проходах следует включить буфер глубины, но на втором нужно также выключить запись в буфер глубины.

Другой вариант применения заключается в указании тех мест изображения, которые не следует отображать, с помощью карт текстур. Благодаря этому будет возможно видеть какие-то части изображения. Для тех участков, которые должны быть видимы, необходимо установить значение альфа-компонента в шаблоне равным 0.0 и 1.0 в противном случае. Такая «переводная картинка» (decal) имеет просвечивающие части, и значения глубины не будут в отношении их эффективны.

### Тест трафарета

Тест трафарета имеет место только при наличии буфера трафарета, иначе он всегда пропускается. При выводе по шаблону эталонное значение для пиксела сравнивается со значением, хранимым в буфере трафарета. В зависимости от результатов сравнения, значение в буфере изменяется. Возможно выбрать функцию сравнения и образцовое значение, а также модифицировать их посредством команд glStencilFunc() и glStencilOp().

void glStencilFunc(GLenum func, Glint ref, GLuint mask)

Устанавливает функцию сравнения (func), эталонное значение (ref) и маску (mask) для их использования в тесте трафарета. Эталонное значение сравнивается со значением из буфера маски заданной функцией, но делается это только для тех битов, для которых биты в маске выставлены в 1. Функция может быть одной из следующих: GL\_NEVER, GL\_ALWAYS, GL\_LESS, GL\_LEQUAL, GL\_EQUAL, GL\_GEQUAL, GL\_GREATER или GL\_NOTEQUAL. Например, если это GL\_LESS, фрагмент считается успешно прошедшим тест, когда значение *ref* меньше соответствующей величины в буфере трафарета. Если буфер трафарета содержит *s* битовых плоскостей, к младшим *s* битам маски, значениям из буфера трафарета и эталонному значению применяется поразрядное И до того, как будет выполнено сравнение. Все маскируемые значения интерпретируются как неотрицательные числа. Тест трафарета включается и выключается указанием константы GL\_STENCIL\_TEST в вызовах glEnable() и glDisable(). По умолчанию функция сравнения (func) – GL\_ALWAYS, *ref* – 0, *mask* – единицы, а обработка по шаблону неактивна.

void glStencilOp(GLenum *fail*, GLenum *zfail*, GLenum *zpass*)

Задаёт, как изменяются данные в буфере трафарета по результату теста трафарета. В качестве аргументов выступают три функции – fail, zfail и zpass, которые могут быть указаны константами GL\_KEEP, GL\_ZERO, GL\_REPLACE, GL\_INCR, GL\_INCR\_WRAP, GL\_DECR, GL\_DECR\_WRAP или GL\_INVERT. Они соответствуют сохранению текущего значения, замене его нулём, замене на эталонное значение, инкременту с насыщением, инкременту без насыщения, декременту с насыщением и без него и побитовому инвертированию. Результат инкрементирования и декрементирования ограничивается диапазоном от нуля до максимально допустимого беззнакового целого числа ( для буфера трафарета с s битовыми плоскостями).

Функция *fail* применяется к фрагментам, потерпевшим неудачу в ходе теста трафарета; для выдержавших испытание фрагментов срабатывает функция *zfail* в случае, если они не прошли тест глубины, и *zpass* – если они преодолели тест глубины или если он не выполнялся. По умолчанию всем трём аргументам присваивается операция GL\_KEEP.

«С насыщением» означает, что значение маски ограничивается неким экстремальным уровнем. Если попытаться уменьшать нуль с насыщением, он так и останется нулём. «Без насыщения» следует понимать так, что при выходе за предел допустимого диапазона значения берутся с другого его конца. Так, при попытке уменьшать нуль без насыщения, значение маски станет максимально возможным беззнаковым целым числом.

### Тест глубины

Буфер глубины хранит для каждого пиксела экрана расстояние между точкой наблюдения и соответствующим местом объекта, ассоциированным с этим пикселом. При рисовании, если тесты глубины успешно пройдены, обрабатываемое значение глубины замещает собой уже находящееся там значение.

В основном буфер глубины используется для исключения невидимых поверхностей. Если новый цвет претендует на пиксел, требования этого объекта удовлетворяются лишь тогда, когда он находится ближе своих предшественников к точке наблюдения. В этом случае после воспроизведения всей сцены на ней остаются только те объекты, которые не заслонены другими. Изначально буфер глубины инициируется значением, соответствующим максимальному удалению от наблюдателя, соответственно, глубина любого объекта заведомо меньше этого значения. Чтобы использовать буфер глубины, нужно разрешить его применение указанием константы GL\_DEPTH\_TEST в вызове glEnable() и помнить, что очистку следует делать перед каждым кадром. Можно также выбрать другую функцию сравнения для теста глубины с помощью команды glDepthFunc().

void glDepthFunc(GLenum *func*)

Устанавливает функцию сравнения для теста глубины. Значение аргумента выбирается из списка: GL\_LESS, GL\_LEQUAL, GL\_EQUAL, GL\_GEQUAL, GL\_GREATER, и GL\_NOTEQUAL. Все поступающие на тест фрагменты проходят его, если их z-значение удовлетворяет заданному соотношению со значением, уже помещённым в буфер глубины. По умолчанию используется функция GL\_LESS, то есть входящий фрагмент признаётся, когда его z-координата меньше ранее занесённой в буфер. В этом случае z-значения представляют собой расстояние от объекта до точки наблюдения, и меньшие величины соответствуют ближе расположенным объектам.

## Смешивание, размытие и логические операции

Поступивший на обработку фрагмент, успешно преодолевший все тесты, допускается к объединению с текущим содержимым буфера цвета одним из нескольких способов. Простейший, и он же по умолчанию, – это перезапись имеющихся значений. Альтернативно, если используется режим RGBA и необходимо сгладить фрагмент или сделать его прозрачным, можно усреднить оба значения – новое и находящееся в буфере, то есть выполнить смешивание (*blending*). Для систем с небольшим количеством доступных цветов можно «размыть» (*dither*) значения цвета, увеличив число доступных полутонов, жертвуя разрешением. На финальной стадии могут быть применены поразрядные логические операции, позволяющие произвольно комбинировать входящий фрагмент и уже записанные в буфер пикселы.

### Смешивание

При смешивании объединяются R-, G-, B- и альфа-компоненты поступивших на обработку фрагментов с ранее обработанными пикселами. Предусмотрены различные операции смешивания, все они зависят от значений альфа-компонентов входящего фрагмента (если он есть) и сохранённых пикселов.

### Размытие

В системах с небольшим количеством битовых плоскостей цвета можно принести разрешение в жертву передаче полутонов (dithering, также размытие, клиширование, смешивание цветов) изображения. Размытие напоминает растрирование (*halftoning*) в полиграфии. Хотя «Нью-Йорк Таймс» печатается только в двух цветах – белом и чёрном, – фотографии на страницах этого издания представлены градациями серого как результат комбинирования чёрных и белых точек. При сравнении газетной фотографии (не в оттенках серого) с её оригиналом (в полутонах) наблюдается потеря пространственного разрешения. Аналогично, системы с поддержкой малого числа битовых плоскостей могут воспользоваться красными, зелёными и синими цветами соседних пикселов для имитации более широкого диапазона цветов.

Операция размытия является аппаратно-зависимой; все реализации OpenGL могут её включить и выключить. Фактически, на иных компьютерах она вообще ни к чему не приводит, поскольку обычно ощущается на машинах с очень точным воспроизведением цвета. Чтобы включить и выключить размытие, задайте параметр GL\_DITHER для команд glEnable() и glDisable(). По умолчанию функция размытия не действует.

Размытие применяется как в RGBA-режиме, так и в режиме индексации цвета. Цвета и цветовые индексы чередуются зависимым от аппаратной части способом, путём выбора из двух ближайших. Например, при индексации цвета, если размытие активно и индекс равен 4.4, из всех пикселов 60% могут быть нарисованы с индексом 4 и 40% – с индексом 5. В режиме RGBA размытие производится раздельно для каждого компонента (включая альфа-канал). Чтобы использовать размытие в режиме индексации цвета, нужно упорядочить цвета в цветовой карте по линейному закону; во избежание некорректного результата.

### Логические операции

Последними над фрагментом выполняются логические операции, как то: ИЛИ (OR), исключающее ИЛИ (XOR) или INVERT. Они применимы к значениям поступившего на обработку фрагмента (источнику) и/или к текущим значениям в буфере цвета (приёмник). Такие операции особенно полезны на компьютерах, оперирующих блоками битов, где первичной графической операцией является копирование прямоугольника с данными из одного места окна в другое, или из окна в оперативную память, или из памяти в окно. Обычно данные при копировании не записываются непосредственно в память, что позволяет выполнить над ними и уже имеющимися данными произвольные логические операции; при этом в приёмник попадает результат операции.

Поскольку этот процесс может быть реализован достаточно «дёшево» на аппаратном уровне, во многие машины логические операции заложены в «железо». Так, XOR может быть использована для стирания-восстановления изображения; повторное применение XOR в том же месте приводит к восстановлению изначального экрана. Как другой пример можно привести интерпретацию индексов цвета в качестве битовых шаблонов. Тогда составное изображение можно комбинировать из слоёв, используя ограничивающие маски записи применительно к различным битовым плоскостям, и модифицировать данные слоёв посредством логических операций.

Включается и выключается режим логических операций заданием константы GL\_INDEX\_LOGIC\_OP или GL\_COLOR\_LOGIC\_OP или при вызове команд glEnable() и glDisable() для режимов индексации цвета или RGBA соответственно. Имеется выбор из 16 логических операций с помощью glLogicOp() или просто применение значение по умолчанию – GL\_COPY.

void glLogicOp(GLenum *opcode*)

Выбирает логическую операцию для применения её к входящему фрагменту (источнику) и пикселу, ранее сохранённому в буфере цвета (приёмнику). В таблице 3 представлены допустимые значения для аргумента кода операции (*opcode*) и соответствующие объяснения (здесь *s* – источник заявок и *d* – приёмник). Значение по умолчанию – GL\_COPY.

|  |  |  |  |
| --- | --- | --- | --- |
| Параметр | Операция | Параметр | Операция |
| GL\_CLEAR | 0 | GL\_AND |  |
| GL\_COPY | s | GL\_OR |  |
| GL\_NOOP | d | GL\_NAND |  |
| GL\_SET | 1 | GL\_NOR |  |
| GL\_COPY\_INVERTED |  | GL\_XOR |  |
| GL\_INVERT |  | GL\_EQUIV |  |
| GL\_AND\_REVERSE |  | GL\_AND\_INVERTED |  |
| GL\_OR\_REVERSE |  | GL\_OR\_INVERVED |  |

*Таблица 3. Логические операции.*

### Буфер накопления

Буфер накопления может быть использован для таких целей, как сглаживание сцены, размытие при движении, имитация фотографической глубины резкости и вычисление результирующих теней от множества источников света. Есть и другие применения, причём в сочетании с другими буферами.

Операции рисования OpenGL не записывают данные напрямую в буфер накопления. Обычно генерируется серия изображений в одном из стандартных буферов цвета, и эти изображения передаются вместе, за один ход, в буфер накопления. Когда накапливать больше нечего, результат копируется назад в буфер цвета для просмотра. С целью уменьшить ошибку округления, буфер накопления может иметь более высокую точность (больше битов на цвет), чем стандартные буфера цвета. Воспроизведение сцены за несколько проходов длиннее одноразового вывода по времени, но даёт более качественные результаты. Решение, что предпочесть в конкретной программе – качество или скорость, – во власти разработчика.

Возможно использовать буфер накопления подобно тому, как фотограф накладывает одну экспозицию на другую, не переводя кадр. При этом движущийся объект получается размытым. Неудивительно, что компьютер способен на большее, чем фотограф со своей камерой. Например, компьютер обладает всеми возможностями контроля над точкой наблюдения, а фотограф не застрахован от случайной встряски камеры.

void glAccum(GLenum *op*, GLfloat *value*)

Управляет буфером накопления. Аргумент op предназначен для выбора операции, а value – используемое во время операции число. Допустимые операции – это GL\_ACCUM, GL\_LOAD, GL\_RETURN, GL\_ADD и GL\_MULT:

* GL\_ACCUM читает каждый пиксел из буфера, выбранного для чтения командой glReadBuffer(), умножает значения R-, G-, B- и альфа-компонентов на *value* и добавляет к результату значения из буфера накопления.
* GL\_LOAD действует аналогично GL\_ACCUM, за исключением того, значения замещают содержимое буфера накопления, а не добавляются к ним.
* GL\_RETURN выбирает значения из буфера накопления, умножает их на *value*, и помещает результат в буфере (буферах) цвета, в которых разрешена запись.
* GL\_ADD добавляет value к значению каждого пиксела из буфера накопления, а GL\_MULT – умножает это значение на value; результат заносится в буфер накопления. Для GL\_MULT множитель (*value*) ограничен диапазоном [-1.0, 1.0]. Для GL\_ADD ограничений нет.

### Сглаживание сцены

Выполняя сглаживание сцены, сначала необходимо очистить буфер накопления и разрешить применение переднего буфера для чтения и записи. После этого несколько раз, например n, необходимо повторить код, чуть-чуть смещающий (*jittering* – дрожание, перемещение изображения в близко расположенную позицию) изображение и отрисовывающий его, накапливая данные командой

glAccum(GL\_ACCUM, 1.0 / *n*)

и последней вызывая

glAccum(GL\_RETURN, 1.0)

Стоит заметить, что этот метод несколько ускорится, если на первом проходе использовать параметр GL\_LOAD и опускать очистку буфера накопления. Здесь изображение рисуется n раз до момента его последнего вывода. Если необходимо избежать показ пользователю промежуточных изображений, необходимо нарисовать в невидимый буфер цвета, накапливайте из него и используйте константу GL\_RETURN для рисования в отображаемый буфер (или в фоновый буфер для последующего обмена его содержимого с передним).

Другой вариант – «обернуть» это в пользовательский интерфейс, показывающий, насколько улучшается изображение после каждого момента накопления, и позволяющий пользователю прервать процесс при достижении изображением необходимого качества. Для этого в цикле, последовательно рисующем изображение, вызывайте glAccum() с параметром GL\_RETURN после каждого акта накопления, задавая числа 16.0/1.0, 16.0/2.0, 16.0/3.0 и т. д. в качестве второго аргумента. Здесь после первого прохода показывается 1/16 конечного изображения, после двух – 2/16 и т. д. После GL\_RETURN должен проверить, нет ли у пользователя желания прервать процесс. Этот интерфейс немного медленнее, поскольку результирующее изображение должно быть скопировано в видимый буфер после каждого прохода.

Выбирая n, необходимо найти компромисс между скоростью (чем большее количество раз рисуется сцена, тем длиннее становится весь процесс) и качеством (чем большее число раз рисуется сцена, тем более удовлетворительней она выглядит, пока не выработана разрешающая способность буфера).

### Глубина резкости

На фотографии, полученной с помощью фотоаппарата, в фокус попадают только элементы, принадлежащие плоскости, удалённой на определённое расстояние от плёнки. С удалением элемента от этой плоскости его чёткость снижается. Глубиной резкости (depth of field) в терминах фотографии называется область рядом с фокальной плоскостью, где потерей чёткостью можно пренебречь.

В нормальных условиях всё, что рисует OpenGL, всегда находится «в фокусе». Буфер накопления может быть использован для приближения к картинке, которую возможно увидеть при изображении, сформированным фотоаппаратом, где элементы тем больше размыты, чем дальше они находятся от фокальной плоскости. Это не точное моделирование съёмки фотоаппаратом, а приближение к нему.

Чтобы достигнуть этого результата, необходимо многократно повторить рисование сцены, вызывая glFrustrum() с различными значениями аргументов. Необходимо подобрать аргументы так, чтобы позиция точки наблюдения слегка отклонялась от её истинной позиции, и чтобы каждый конус угла зрения проецировался в общий прямоугольник, принадлежащий фокальной плоскости. Результаты по всем вариантам воспроизведения сцены нужно усреднить обычным образом через буфер накопления.

# Содержательная постановка задачи

В среде программирования Visual Studio 2015 (или выше) с помощью платформонезависимой графической библиотеки OpenGL написать программу, которая будет содержать средства для работы с буфером кадров.

При работе с буфером кадров продемонстрировать работу со следующими элементами:

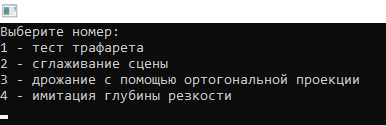
* Тест трафарета
* Сглаживание сцены
* Дрожание с помощью ортогональной проекции
* Имитация глубины резкости

При запуске программы следует предоставить пользователю выбор, какой из вышеперечисленных примеров следует продемонстрировать на экране. Так, по нажатию клавиши «1» должен демонстрироваться тест трафарета, по нажатию клавиши «2» должно демонстрироваться сглаживание сцены и т. д.

Также следует выводить результат работы определённого элемента программы в новом окне.

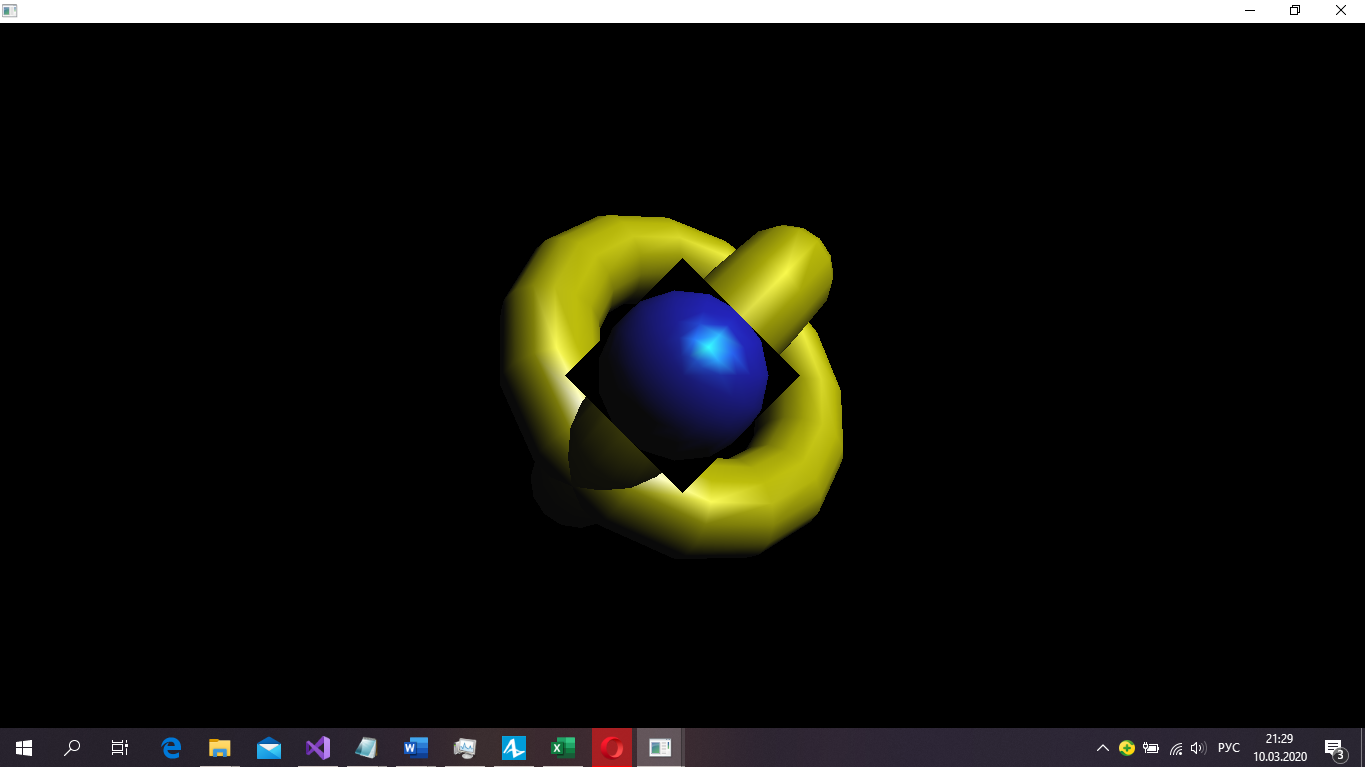
# Демонстрация работы программы

При запуске приложения открывается начальное окно, в котором пользователю предлагается выбрать один из четырёх элементов для демонстрации работы приложения (рис. 2).



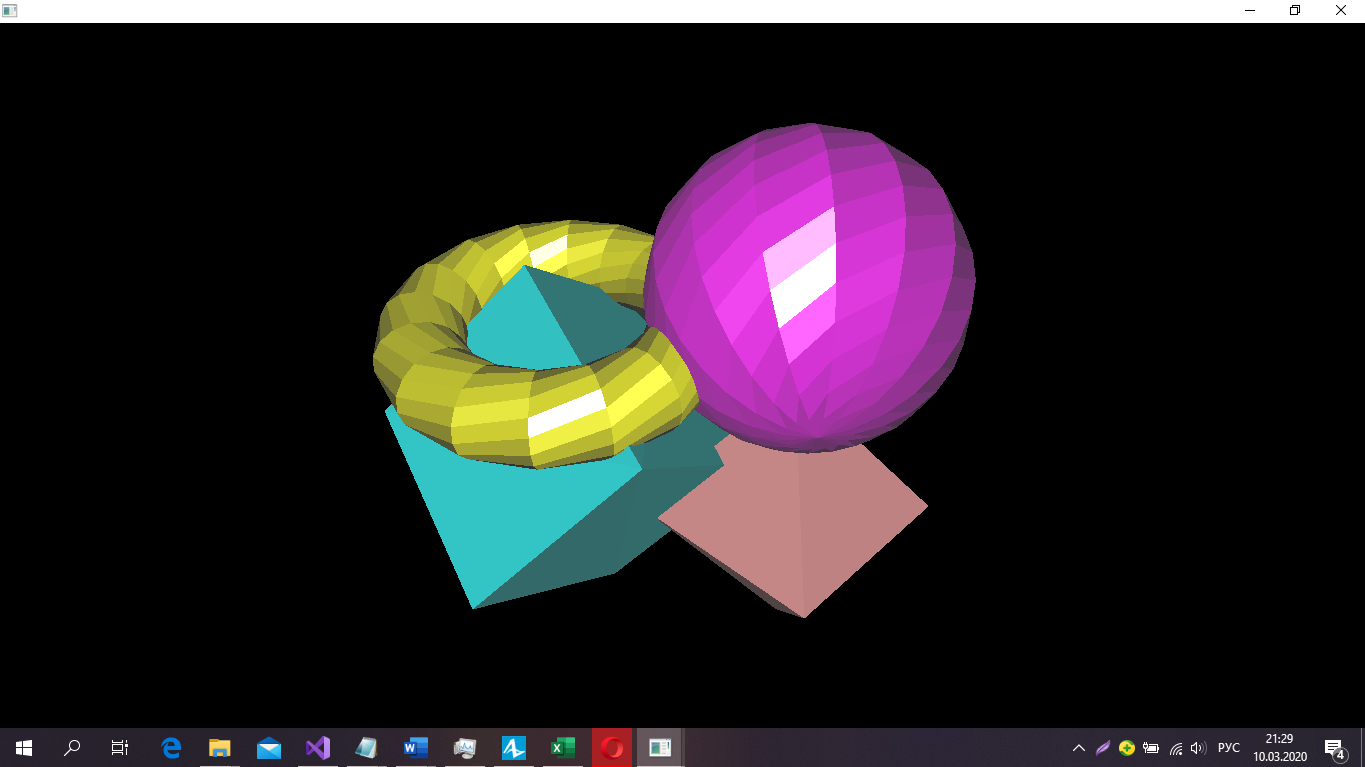
*Рис. 2. Главное окно программы.*

При нажатии на клавишу «1» на экране демонстрируется тест трафарета (рис. 3). В новом окне рисуются два тора с ромбовидным контуром в центре сцены. В пределах этого контура размещается сфера. Здесь отображение в буфер трафарета имеет место только в момент перерисовки окна, поэтому буфер цвета очищается после создания трафарета.



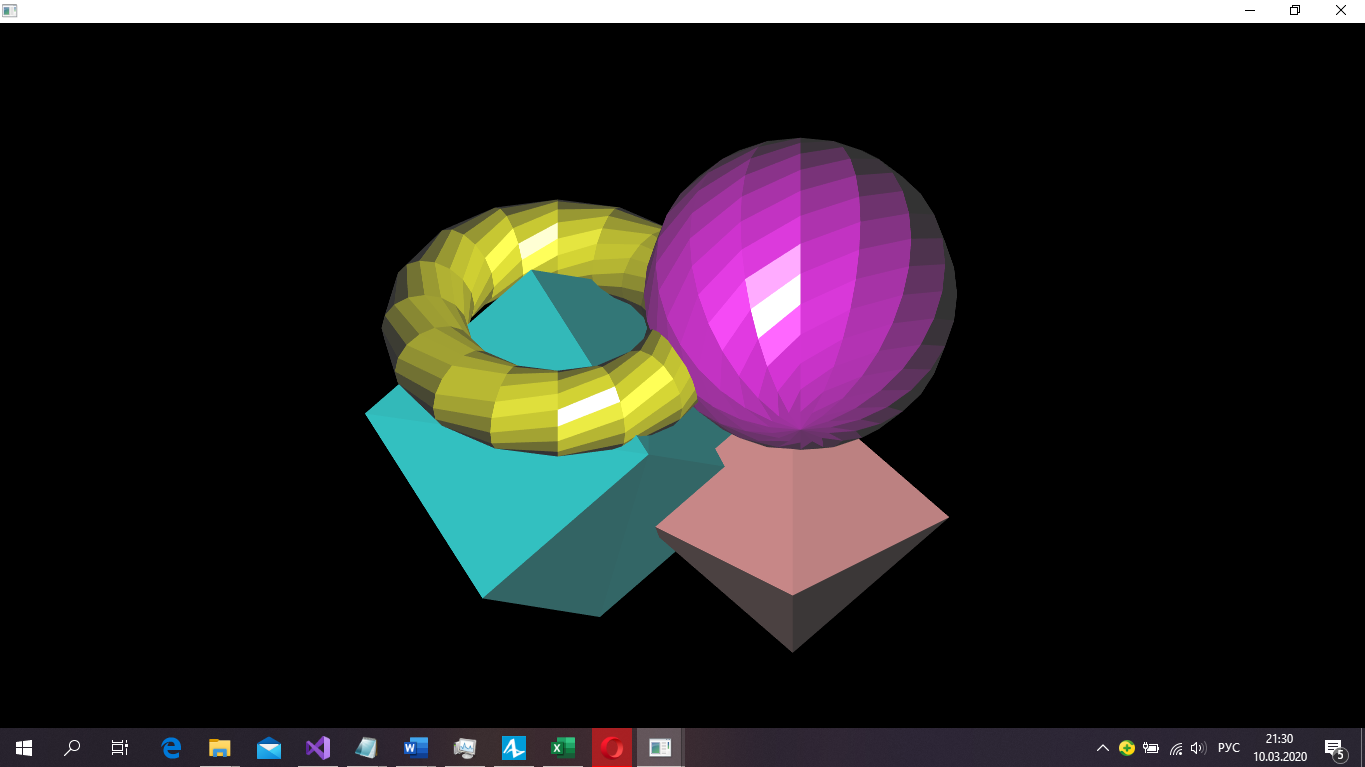
*Рис. 3. Тест трафарета.*

При нажатии на клавишу «2» на экране демонстрируется результат сглаживания сцены (рис. 4). В код для отрисовки данных объектов включены две процедуры для «дрожания» и сглаживания сцены, которые являются полезными: accPerspective() и accFrustum(). Процедура accPerspective() используется вместо команды gluPerspective(), первые четыре аргумента обоих процедур одинаковые. Чтобы заставить «дрожать» конус угла зрения, необходимо указать x и y в качестве значений пятого и шестого аргументов accPerspective(). Также необходимо присвоить значение 0.0 седьмому и восьмому аргументам accPerspective() и ненулевое значение девятому (с целью упреждения деления на нуль в accPerspective()). Эти последние три параметра используются для получения эффекта резко очерченного пространства (глубины резкости).



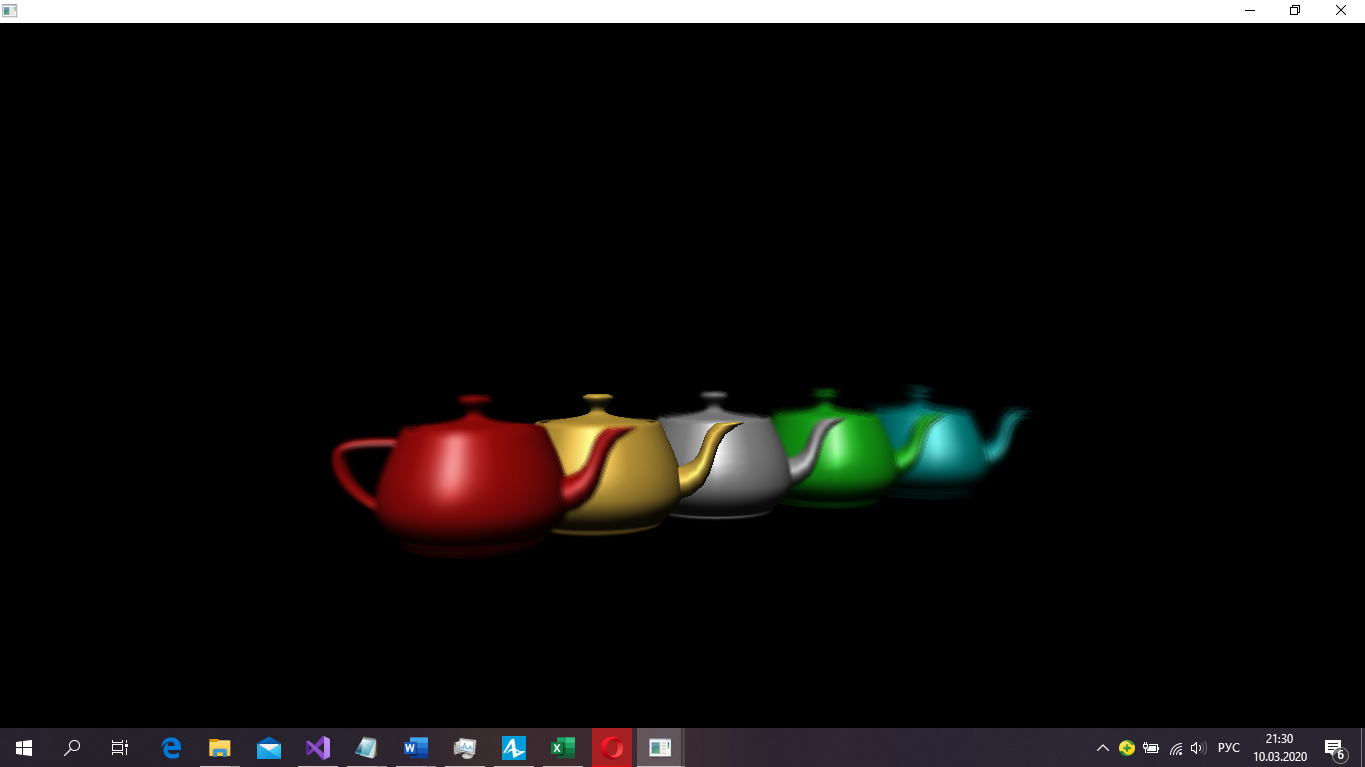
*Рис. 4. Сглаживание сцены.*

При нажатии на клавишу «3» на экране демонстрируется результат «дрожания» с помощью ортогональной проекции (рис. 5). Стоит отметить, что нельзя использовать перспективную проекцию для сглаживания сцены. Ступенчатость сцены можно устранять с ортогональной проекцией, вызывая glTranslate\*() для получения «дрожания». Также стоит иметь в виду, что команда glTranslate\*() оперирует мировыми (world) координатами, а в данном задании гораздо удобнее использовать экранные координаты. Таким образом, необходимо самостоятельно произвести соответствующее преобразование, разделив высоту и ширину в мировых координатах на размеры области просмотра. Затем необходимо умножить значение в мировых координатах на интервал дрожания, чтобы «предсказать» максимальную величину сдвига – меньше 1 пиксела.



*Рис. 5. «Дрожание» с помощью ортогональной проекции.*

При нажатии на клавишу «4» на экране демонстрируется результат имитации глубины резкости. На рисунке 6 показано изображение пяти чайников, сформированное с имитацией глубины резкости. Второй слева чайник находится в фокусе, а другие теряют чёткость при удалении от фокальной плоскости. Данная сцена рисуется 8 раз с небольшим отклонением, вызовом accPerspective(). Чтобы получить эффект глубины резкости, положение фокальной плоскости z должно быть неизменным. Оно задаётся девятым (последним) аргументом accPerspective(), и в текущем случае равно 5.0. Уровень размытия определяется константой – произведением смещений «дрожания» x и y (седьмой и восьмой параметры accPerspective()). Её значение не выводится теоретически, а подбирается экспериментально на основании значений глубин резкости изображаемого пространства. Стоит отметить, что в данном случае пятый и шестой аргументы accPerspective() равны 0, что означает отсутствие сглаживание сцены.



*Рис. 6. Имитация глубины резкости.*

# Выводы

В ходе данного домашнего задания был проведён анализ литературы на тему «Буфер кадров». В ходе данного анализа были выбраны средства для реализации программы, а также был отобран набор элементов для визуализации. Кроме этого, была написана программа, в которой наглядно демонстрируются результаты работы со следующими элементами буфера кадров:

* Тест трафарета
* Сглаживание сцены
* Дрожание с помощью ортогональной проекции
* Имитация глубины резкости

В ходе данного домашнего задания также были подробно описаны алгоритмы работы программы с описанием используемых функций. Исходя из поставленных целей и достигнутых результатов, домашняя работа полностью выполнена.

# Список используемой литературы

* М. Ву, Т. Девис, Дж. Нейдер, Д. Шрайнер. OpenGL. Руководство по программированию. 4-е издание. Стр. 386-415.