Министерство образования и науки Российской Федерации КАЗАНСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. А.Н.Туполева - КАИ

# М.В. МЕДВЕДЕВ

**РАБОТА С РАСТРОВЫМИ ИЗОБРАЖЕНИЯМИ В OPENGL**

*Методическое пособие*

*к лабораторным работам по курсу*

*«Компьютерная геометрия и графика»*

# Казань 2014

**РАБОТА С РАСТРОВЫМИ ИЗОБРАЖЕНИЯМИ В OPENGL**

Растровое изображение — изображение, представляющее собой сетку пикселей или цветных точек на отображающих устройствах. Важными характеристиками изображения являются количество пикселей (размер изображения по ширине и высоте в пикселях) и цветовое пространство (цветовая модель), используемое для представления цвета каждого пикселя. Обычно источником растровых изображений в OpenGL являются буферы цветов.

## Чтение, запись и копирование пикселей

* 1. ***Чтение пикселей***

Функция **glReadPixels**() читает прямоугольный массив пикселей из буфера кадров и сохраняет данные в оперативной памяти.

void glReadPixels(GLint x, Glint y, GLsizei width, GLsizei height, GLenum format,GLenum type, GLvoid

\*pixels);

Функция читает из буфера кадров пиксельный прямоугольник, левый нижний угол которого находится в позиции (*х*, *у*) в оконных координатах с шириной *width* и высотой *height,* и сохраняет данные в массиве *pixels*. Аргумент *format* задает формат читаемого элемента (индекс цвета или компоненты R, G, В, А, в соответствии с табл. 1). Аргумент *type* уточняет тип данных каждого элемента (табл. 2).

Табл. 1. Форматы пикселей для команд *glReadPixels*() и *glDrawPixels*().

|  |  |
| --- | --- |
| **Формат** | **Описание** |
| GL\_COLOR\_INDEX | Индекс цвета |
| GL\_RGB | Красный компонент цвета с сопутствующими зеленым  и синим компонентами |

|  |  |
| --- | --- |
| GL\_RGBA | Красный компонент цвета с сопутствующими зеленым,  синим и альфа-компонентами |
| GL\_BGR\_EXT | Синий компонент цвета с сопутствующими зеленым  и красным компонентами |
| GL\_BGRA\_EXT | Синий компонент цвета с сопутствующими зеленым,  красным и альфа-компонентами |
| GL\_RED | Красный компонент цвета |
| GL\_GREEN | Зеленый компонент цвета |
| GL\_BLUE | Синий компонент цвета |
| GL\_ALPHA | Альфа-компонент цвета |
| GL\_LUMINANCE | Компонент яркости |
| GL\_LUMINANCE\_ALPHA | Компонент яркости с сопутствующим альфа-компонентом |
| GL\_STENCIL\_INDEX | Индекс трафарета |
| GL\_DEPTH\_COMPONENT | Компонент глубины |

Табл. 2. Типы данных для команд *glReadPixels*() и *glDrawPixels*().

|  |  |
| --- | --- |
| **Константа типа данных** | **Тип данных** |
| GL\_UNSIGNED\_BYTE | 8-разрядное целое число без знака |
| GL\_BYTE | 8-разрядное целое число со знаком |
| GL\_UNSIGNED\_SHORT | 16-разрядное целое число без знака |
| GL\_SHORT | 16-разрядное целое число со знаком |
| GL\_UNSIGNED\_INT | 32-разрядное целое число без знака |
| GL\_INT | 32-разрядное целое число со знаком |
| GL\_FLOAT | Число с плавающей запятой одинарной точности |

Если используется команда *glReadPixels*() для получения RGBA- данных или определенных индексом данных, требуется уточнить буфер для доступа. Например, если окно имеет двойную буферизацию, нужно указать, следует читать данные из буфера переднего плана или заднего плана. Для этого используется команда *glReadBuffer*().

void glReadBuffer(GLenum mode);

Функция выбирает буфер цвета, активируемый как источник для чтения пикселей командами *glReadPixels*(), *glCopyPixels*(), и отключает буферы,

включенные предшествующими вызовами *glReadBuffer*(). Значение режима

*mode* может быть одним из указанных в табл. 3.

Табл. 3. Некоторые типы буферов OpenGL.

|  |  |
| --- | --- |
| **Константа типа буфера** | **Тип буфера** |
| GL\_FRONT | Передний буфер |
| GL\_FRONT\_LEFT | Передний левый буфер |
| GL\_BACK | Задний буфер |
| GL\_FRONT\_RIGHT | Передний правый буфер |
| GL\_LEFT | Левый буфер |
| GL\_BACK\_LEFT | Задний левый буфер |
| GL\_RIGHT | Правый буфер |
| GL\_BACK\_RIGHT | Задний правый буфер |

Буферы цвета – это буферы, 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. По умолчанию аргумент *mode* устанавливается в значение *GL\_FRONT* для контекстов с однократной буферизацией и в значение *GL\_BACK* для двойной буферизации.

## Запись пикселей

Отображение пикселей осуществляется при помощи функции

# glDrawPixels().

void glDrawPixels(GLsizei width, Glsizei height, GLenum format, GLenum type, const GLvoid \*pixels)

Функция отображает прямоугольник из пикселей, с шириной *width* и высотой *height*. Левый нижний угол прямоугольника соответствует текущей позиции растра. Аргументы *format* и *type* несут ту же нагрузку, что и для функции *glReadPixels*() (см. табл. 1 и 2). Массив *pixels* содержит данные для отрисовки. Если текущая растровая позиция некорректна, никакого вывода не производится.

Функция ***glRasterPos*\* ()** задает текущую позицию растра, используемую *glDrawPixels*() и *glCopyPixels*(). Эти координаты преобразуются конвейером геометрической обработки. Вследствие различного формата представления буфера кадров, наличия множества способов хранения информации о пикселях в памяти компьютера и различных преобразований данных, выполняемых во время чтения, записи и копирования возникает множество режимов выполнения команд.

## Копирование пикселей

Функция **glCopyPixels()** копирует прямоугольный массив пикселей из одной части буфера кадров в другую. Эта команда ведет себя подобно последовательности из вызовов *glReadPixels*() и *glDrawPixels*(), но оперативная память не принимает участия в передаче данных.

void glCopyPixels(GLint х, GLint у, GLsizei width, GLsizei height, Glenum buffer)

Функция копирует прямоугольную область внутри буфера кадров с позицией левого нижнего угла (*х*, *у*), шириной *width* и высотой *height*. Левый нижний угол целевого прямоугольника совпадает с текущей позицией растра. Аргумент *buffer* (*GL\_COLOR*, либо *GL\_STENCIL*, либо *GL\_DEPTH*) указывает тип используемого буфера. Функция *glCopyPixeis*() ведет себя аналогично последовательности из *glReadPixels*() и *glDrawPixels*(), с промежуточным преобразованием константы *buffer* в аргумент *format.*

Если *buffer* равен *GL\_DEPTH* или *GL\_STENCIL*, используется *GL\_DEPTH\_COMPONENT* или *GL\_STENCIL\_INDEX* соответственно. Если указано значение *GL\_COLOR*, используется *GL\_RGBA* или *GL\_COLOR\_INDEX*, в зависимости от режима цвета.

В аргументах *format* пли *data* при использовании *glCopyPixeis*() нет необходимости, так как данные никогда не поступают через оперативную память. Исходный и целевой буферы для *glCopyPixei*() указываются с помощью *glReadBuffer*() и *glDrawBuffer*() соответственно.

## Упаковка и распаковка пикселей

Изображение сохраняется в памяти в нескольких элементах. Данные могут представляться просто индексом цвета или значением яркости (то есть средневзвешенным значением красного, зеленого и синего компонентов) или же быть выражены красным, зеленым, синим и альфа-компонентом (RGBA) для каждого пикселя. Возможное размещение данных пикселей, то есть формат, определяется количеством элементов и их порядком. Упаковка и распаковка определяет, в каком направлении следуют данные на запись или чтение из оперативной памяти.

Некоторые элементы (такие как индексы цвета или трафарета) являются целочисленными, а другие (RGBА или глубины) — значениями с плавающей запятой (вещественными), обычно в диапазоне между 0 и 1. Вещественные компоненты обычно сохраняются в буфере кадров с более низким разрешением, чем позволяет число с плавающей запятой (например, компонентам цвета может отводиться по 8 бит). Точное количество битов, используемых для представления компонентов, зависит от особенностей аппаратной части. Таким образом, сохранение каждого компонента в виде 32- разрядного вещественного числа часто теряет смысл, тем более что изображения могут содержать миллионы пикселей.

Элементы могут сохраняться в памяти с различными типами данных, от 8-разрядного байта до 32-разрядного целого или числа с плавающей

запятой. OpenGL явно определяет преобразование каждого компонента в каждом формате для каждого допустимого типа данных.

Данные изображения обычно сохраняются в оперативной памяти в виде прямоугольных (двух- или трехмерных) массивов. Различные компьютеры имеют различный порядок следования байтов (рис. 1). Аппаратное обеспечение некоторых машин более эффективно работает с данными, выровненными по 2-, 4- или 8-байтовой границе оперативной памяти.

7 6 5By4t3e 2 1

0

7 6 5By4t3e 2 1

0

Short (byte 1)

Short (byte 0)

15 14 13 12 11 10 9 8

Short (byte 1)

Short (byte 0)

7 6 5 4 3 2 1 0

7 6 5 4 3 2 1 0 15 14 13 12 11 10 9 8

31 30 29 28 27 26 25

|  |  |  |  |
| --- | --- | --- | --- |
| Integer (byte 0) | Integer (byte 1) | Integer (byte 2) | Integer (byte 3) |

24

23 22 21 20 19 18 17 16

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

|  |  |  |  |
| --- | --- | --- | --- |
| Integer (byte 3) | Integer (byte 2) | Integer (byte 1) | Integer (byte 0) |

723 3126152 3215014210432189312178212116711210560

Рис. 1. Эффект для перестановки байтов для типов *Byte*, *Short* и *Integer*

Все поддерживаемые OpenGL режимы сохранения пикселей управляются командой ***glPixelStore\** ()**. Как правило, выполняется несколько последовательных вызовов этой команды с целью установить значения нескольких параметров.

void glPixelStore{if}(GLenum pname, TYPE param);

Функция устанавливает режимы сохранения пикселей применительно к операциям *glDrawPixels*(), *glReadPixels*() и др. Возможные имена параметра *pname* представлены в табл. 4 вместе с типами данных, начальными значениями и допустимыми диапазонами значений. Параметры *GL\_UNPACK\** управляют тем, как выполняется распаковка данных из памяти. Параметры *GL\_PACK\** аналогично отвечают за упаковку в память.

Табл. 4. Параметры команды *glPixelStore*() (при работе в 2D).

|  |  |  |  |
| --- | --- | --- | --- |
| **Название параметра** | **Тип** | **Значение**  **инициализации** | **Допустимый диапазон** |
| GL\_UNPACK\_SWAP\_BYTES,  GL\_PACK\_SWAP\_BYTES | GLboolean | FALSE | TRUE/FALSE |
| GL\_UNPACK\_LSB\_FIRST,  GL\_PACK\_LSB\_FIRST | GLboolean | FALSE | TRUE/FALSE |
| GL\_UNPACK\_ROW\_LENGTH,  GL\_PACK\_ROW\_LENGTH | GLint | 0 | Любое неотрицательное  целое число |
| GL\_UNPACK\_SKIP\_ROWS,  GL\_PACK\_SKIP\_ROWS | GLint | 0 | Любое неотрицательное  целое число |
| GL\_UNPACK\_SKIP\_PIXELS,  GL\_PACK\_SKIP\_PIXELS | GLint | 0 | Любое неотрицательное  целое число |
| GL\_UNPACK\_ALIGNMENT,  GL\_PACK\_ALIGNMENT | GLint | 4 | 1,2,4,8 |

***Примечание.*** *Поскольку соответствующие упаковке и распаковке параметры имеют одинаковые значения, они обсуждаются далее вместе и упоминаются без префикса GL\_PACK или GL\_UNPACK. Например, \*SWAP\_BYTES подразумевает как GL\_PACK\_SWAP\_BYTES, так и GL\_UNPACK\_SWAP\_BYTES.*

Если параметр *\*SWAP\_BYTES* равен *FALSE* (по умолчанию), действует

«родной»

для клиента OpenGL порядок следования байтов в памяти, и в противном случае - обратный. Изменение порядка следования применимо к элементам любого размера, но имеет смысл только для многобайтовых элементов.

Эффект от перестановки байтов может различаться от реализации OpenGL к реализации. Перестановка не влияет на однобайтовые данные. Пока приложение OpenGL работает с изображениями, не разделяя их с

другими машинами, можно игнорировать аспект порядка следования байтов. Но если оно должно воспроизвести изображение OpenGL, созданное на другой машине, с иным расположением байтов в слове, нужно устанавливать параметр *\*SWAP\_BYTES*. Тем не менее параметр *\*SWAP\_BYTES* не позволяет переупорядочивать элементы (например, переставлять местами зеленый и красный). Параметр *\*LSB\_FIRST* применяется только при отображении или чтении одноразрядных изображений или битовых образов, где одному пикселю соответствует один бит. Если значение *\*LSB\_FIRST* равно *FALSE* (по умолчанию), биты каждого байта отсчитываются от позиции самого старшего бита; иначе подразумевается противоположный порядок. Например, если

*\*LSB\_FIRST* равно *FALSE*, а интересующий байт — 0x31, биты следуют в последовательности {0, 0, 1, 1, 0, 0, 0, 1}. Если *\*LSB\_FIRST* равно *TRUE*, эта

последовательность выглядит так: {1, 0, 0, 0,1,1, 0,0}.

Иногда нужно отрисовать или прочитать только прямоугольник, являющийся частью изображения, сохраненного в памяти. В таком случае требуется указать действительную длину (в пикселях) большего прямоугольника с помощью параметра *\*ROW\_LENGTH*. Если значение

*\*ROW\_LENGTH* нулевое (по умолчанию), длина ряда приравнивается ширине, указанной в *glReadPixels*(), *glDrawPixels*() или *glCopyPixels*(). Также требуется задать количество рядов и пикселей в ряду, пропускаемых перед копированием вложенного прямоугольника. Эти числа задаются параметрами

*\*SKIP\_R0WS* и *\*SKIP\_PIXELS*, как показано на рис. 2. По умолчанию оба значения равны 0, то есть обработка осуществляется от левого нижнего угла.

*\*ROW\_LENGTH*

*\*SKIP\_PIXELS*

*\*SKIP\_ROWS*

Рис. 2. Параметры *\*SKIP\_ROWS*, *\*SKIP\_PIXELS* и *\*ROW\_LENGTH.*

Часто аппаратные средства компьютеров оптимизированы под выровненные в памяти данные. Например, если длина слова в компьютере 32 бита, оборудование с большей эффективностью будет обращаться к данным, выровненным по 32-разрядной границе, адреса которых обычно кратны 4. Аналогично, 64-битовые архитектуры могут работать лучше, когда данные выровнены по 8-байтовым границам. На некоторых компьютерах, тем не менее, выравнивание не играет роли.

Предположим, что ваша машина лучше справляется с пикселями, выровненными по 4-байтовой границе. Изображения хранятся с учетом этого фактора, и каждый ряд начинается с адреса, кратного 4 байтам. Если изображение имеет ширину 5 пикселей и для каждого пикселя отводится по байту на компонент (красный, зеленый и синий), то вся строка займет 5x3=15 байт. Для достижения максимальной эффективности требуется дополнительное выделение еще одного байта под каждую строку. Если данные выровнены подобным образом, следует задать параметр

*\*ALIGNMENT* (4 в данном случае).

Если значение *\*ALIGNMENT* установлено в 1, задействуется каждый байт. Если оно равно 2, при необходимости пропускается один байт в конце строки для того, чтобы следующая строка начиналась с четного адреса. В случае битовых образов (или одноразрядных изображений) выравнивание производится аналогично, с той разницей, что обрабатываются не байты, а биты.

## Операции при передаче пикселей

Во время передачи данных из памяти в буфер кадров (или наоборот) OpenGL может производить над этими данными некоторые действия. Могут создаваться карты (maps) для выполнения произвольных преобразований над индексом цвета или компонентами цвета. Такие преобразования выполняются во время передачи пикселей в (из) буфер кадров и соответственно называются операциями передачи пикселей (pixel-transfer

operations). Отвечают за такие преобразования команды *glPixelTransfer\**() и

*glPixelMap\**().

void glPixelTransfer{if}(GLenum рnаmе, TYPE раrаm)

Функция устанавливает режимы передачи пикселей. Аргумент *рname* принимает одно из значений, перечисленных в первом столбце табл. 5, а значения аргумента *param* должны быть в допустимом диапазоне, как показано в той же таблице.

Табл. 5. Некоторые параметры команды *glPixelTransfer*\*().

|  |  |  |  |
| --- | --- | --- | --- |
| **Название**  **параметра** | **Тип** | **Значение**  **инициализации** | **Допустимый**  **диапазон** |
| GL\_MAP\_COLOR | GLboolean | FALSE | TRUE/FALSE |
| GL\_INDEX\_SHIFT | GLint | 0 | (-∞, ∞) |
| GL\_INDEX\_OFFSET | GLint | 0 | (-∞, ∞) |
| GL\_RED\_SCALE | GLfloat | 1.0 | (-∞, ∞) |
| GL\_GREEN\_SCALE | GLfloat | 1.0 | (-∞, ∞) |
| GL\_BLUE\_SCALE | GLfloat | 1.0 | (-∞, ∞) |
| GL\_ALPHA\_SCALE | GLfloat | 1.0 | (-∞, ∞) |
| GL\_RED\_BIAS | GLfloat | 0.0 | (-∞, ∞) |
| GL\_GREEN\_BIAS | GLfloat | 0.0 | (-∞, ∞) |
| GL\_BLUE\_BIAS | GLfloat | 0.0 | (-∞, ∞) |
| GL\_ALPHA\_BIAS | GLfloat | 0.0 | (-∞, ∞) |

Если параметр *GL\_MAP\_COLOR* установлен в *TRUE*, доступно табличное преобразование (mapping). Все остальные параметры непосредственно относятся к компонентам пикселей.

Масштабирование (scale) и отклонение (bias) применимы к красному, зеленому, синему, альфа-компонентам и к глубине. Например, можно масштабировать компоненты RGB в процессе сведения их к яркости. Яркость вычисляется как сумма красного, зеленого и синего цветов. Если преобразование из модели RGB к яркости выполняется согласно стандарту

NTSC, нужно устанавливать компоненты в конкретные значения:

*GL\_RED\_SCALE* – 0.3, *GL\_GREEN\_SCALE* – 0.59 и *GL\_BLUE\_SCALE* – 0.11.

## Табличные преобразования пикселей

Все компоненты цвета, индексы цвета и трафарета могут быть модифицированы с помощью таблиц подстановки перед их помещением в видеопамять. Этот процесс называется табличным преобразованием (mapping) и управляется командой *glPixelMap*\*().

void glPixelMap{ui us f}v(GLenum map, GLint mapsize, const TYPE \*values);

Функция загружает пиксельную карту (таблицу) *map*, состоящую из *mapsize* элементов, соответствующих значениям (массива) *values*. Табл. 6 содержит список названий карт и значений; по умолчанию все размеры считаются равными 1, а значения — 0. Размер каждой карты должен быть степенью 2.

Табл. 6. Названия и значения некоторых аргументов команды *glPixelMap*\*()

|  |  |  |
| --- | --- | --- |
| **Название карты** | **Адрес** | **Значение** |
| GL\_PIXEL\_MAP\_I\_TO\_I | Индекс цвета | Индекс цвета |
| GL\_PIXEL\_MAP\_I\_TO\_R | Индекс цвета | R |
| GL\_PIXEL\_MAP\_I\_TO\_G | Индекс цвета | G |
| GL\_PIXEL\_MAP\_I\_TO\_B | Индекс цвета | B |
| GL\_PIXEL\_MAP\_I\_TO\_A | Индекс цвета | A |
| GL\_PIXEL\_MAP\_R\_TO\_R | R | R |
| GL\_PIXEL\_MAP\_G\_TO\_G | G | G |
| GL\_PIXEL\_MAP\_B\_TO\_B | B | B |
| GL\_PIXEL\_MAP\_A\_TO\_A | A | A |

## Растяжение, сжатие и зеркальное отражение изображений

После применения режимов сохранения пикселей и операций при передаче пикселей изображения и битовые образы растеризуются. В простом варианте каждому пикселю изображения соответствует один пиксель экрана. Существует возможность произвольно увеличивать и уменьшать изображение и зеркально отражать его с помощью команды *glPixelZoom*().

void glPixelZoom(GLfloat zoomx, GLfloat zoomy);

Функция задает коэффициенты растяжения и сжатия для операций записи пикселей (*glDrawPixels*() и *glCopyPixels*()) по горизонтали и вертикали. По умолчанию значения *zoomx* и *zoomy* устанавливаются в 1. Во время растеризации каждый пиксель изображения интерпретируется как прямоугольник с размерами *zoomx\*zoomy*, и фрагменты генерируются для всех пикселей, центры которых находятся в пределах этого прямоугольника.

## Создание меню

В библиотеке GLUT имеются функции поддержки работы с всплывающими меню (pop-up menus). Работа с меню в прикладной программе включает несколько операций.

1. Определить функцию с обратным вызовом для обработки выбора пунктов меню при помощи функции:

glutCreateMenu(*имя\_функции\_обратного\_вызова*);

Функция обратного вызова должна иметь тип *void* и единственный входной параметр — идетификатор выбранного пункта меню:

void demomenu(int id);

1. Определить в программе пункты меню при помощи функции:

glutAddMenuEntry("Имя пункта меню", *номер\_пункта\_меню*);

1. Связать меню с определенной кнопкой мыши при помощи функции:

glutAttachMenu(*кнопка\_мыши*);

Ниже приведен пример создания меню из трех пунктов.

glutCreateMenu(demomenu); glutAddMenuEntry("quit",1); glutAddMenuEntry("increase square size", 2); glutAddMenuEntry("decrease square size", 3); glutAttachMenu(GLUT\_RIGHT\_BUTTON);

Библиотека GLUT поддерживает работу и с иерархическими меню. Для этого нужно объявить указатель на подменю и функцию обратного вызова для подменю и добавить их в основное менюследующим образом.

1. Создать указатель на подменю и привязать к нему функцию подменю.

*GLint указатель\_на\_подменю=*

glutCreateMenu(*имя\_функции\_подменю*);

1. Добавить пункты в подменю.
2. Создать основное меню и добавить в него пункты.
3. Одним из пунктов основного меню добавить подменю.

glutAddSubMenu("Имя подменю", *имя\_функции\_подменю*);

Пусть главное меню состоит только из двух пунктов. При выборе первого пункта выполнение программы завершается, а при выборе второго на экран выводится подменю. Пример программного кода для такого случая представлен ниже.

*GLint* submenu = glutCreateMenu(sizemenu); glutAddMenuEntry("increase square size", 2); glutAddMenuEntry("decrease square size", 3); glutCreateMenu(topmenu); glutAddMenuEntry("quit",1); glutAddSubMenu("Resize", submenu);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

## Задания для самостоятельного выполнения

1. Осуществить чтение/запись изображения в формате с типом компоненты цвета *К* (табл. 7) и форматом данных *Ф* (табл. 8). Реализовать вызов этой функции через подменю.
2. Осуществить масштабирование изображения согласно заданию *М*

(табл. 9). Добавить соответствующий пункт меню.

1. Осуществить отображение только тех компонент, которые указаны в задании *Т* (табл. 10) Добавить соответствующий пункт меню.
2. Осуществить инверсию компонент цвета, которые указаны в задании *Т*

(табл. 10). Добавить соответствующий пункт меню.

Варианты заданий указаны в табл. 11. Пример программы обработки растровых изображений в OpenGL приведен в приложении.

Табл. 7.

|  |  |
| --- | --- |
| **Номер задания** | **Формат компонентов цвета** |
| К1 | Красный компонент цвета с сопутствующими зеленым  и синим компонентами |
| К2 | Красный компонент цвета с сопутствующими зеленым,  синим и альфа-компонентами |
| К3 | Синий компонент цвета с сопутствующими зеленым  и красным компонентами |
| К4 | Синий компонент цвета с сопутствующими зеленым,  красным и альфа-компонентами |
| К5 | Компонент яркости |
| К6 | Компонент яркости с сопутствующим альфа-компонентом |

Табл. 8.

|  |  |
| --- | --- |
| **Номер задания** | **Тип данных** |
| Ф1 | 8-разрядное целое число без знака |
| Ф2 | 8-разрядное целое число со знаком |
| Ф3 | 16-разрядное целое число без знака |
| Ф4 | 16-разрядное целое число со знаком |
| Ф5 | 32-разрядное целое число без знака |

|  |  |
| --- | --- |
| Ф6 | 32-разрядное целое число со знаком |
| Ф7 | Число с плавающей запятой одинарной точности |

Табл. 9.

|  |  |
| --- | --- |
| **Номер задания** | **Текст задания** |
| М1 | Отразить изображение относительно оси *Х* |
| М2 | Отразить изображение относительно оси *Y* |
| М3 | Отразить изображение относительно точки (0;0) |
| М4 | Увеличить изображение в 2 раза |
| М5 | Уменьшить изображение в 2 раза |

Табл. 10.

|  |  |
| --- | --- |
| **Номер задания** | **Компоненты цвета для отображения** |
| T1 | R |
| Т2 | G |
| Т3 | B |
| Т4 | R, G |
| Т5 | B, G |
| Т6 | R, B |
| Т7 | B, G |

Табл. 11.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **варианта** | **К** | **М** | **Т** | **Ф** |
| 1 | К1 | М1 | T6 | Ф6 |
| 2 | К2 | М2 | Т5 | Ф2 |
| 3 | К3 | М3 | Т4 | Ф3 |
| 4 | К4 | М4 | Т3 | Ф4 |
| 5 | К5 | М5 | Т2 | Ф5 |
| 6 | К6 | М1 | Т1 | Ф6 |
| 7 | К1 | М2 | T1 | Ф7 |
| 8 | К2 | М3 | Т2 | Ф1 |
| 9 | К3 | М4 | Т3 | Ф2 |
| 10 | К4 | М5 | Т4 | Ф3 |
| 11 | К5 | М1 | Т5 | Ф4 |
| 12 | К6 | М2 | Т6 | Ф5 |

*Приложение.*

#include <glut.h> #include <math.h> #include <glaux.h> #include <stdio.h>

#pragma comment(lib, "glaux.lib")

/\*Структура для хранения заголовка файла изображения\*/ struct Zagolovok

{

GLint shirina; /\*Ширина\*/ GLint vysota; /\*Высота\*/

GLenum formatCveta; /\*Формат представления цвета\*/

GLenum formatKomponenty; /\*Формат данных компоненты цвета\*/ int kol\_voKomponent; /\*Количество компонент цвета\*/

};

/\*Структура для хранения изображения\*/ struct Izobrajenie

{

unsigned char \*pikseli; Zagolovok zagolovok;

};

/\*Функция чтения изображения из фалйа .kai\*/

Izobrajenie \*ChtenieIzobrajeniyaIzFaila(const char \*szFileName)

{

FILE \*pFile; // Указатель файла

Izobrajenie \*iz=(Izobrajenie \*)malloc(sizeof(Izobrajenie)); //Создание структуры для хранения изображения

// Открытие файла pFile=fopen(szFileName, "rb"); if(pFile == NULL) return NULL;

// Считываем заголовок

fread(&(iz->zagolovok), sizeof(Zagolovok), 1, pFile);

//Создание массива для битов

iz->pikseli=(unsigned char \*)malloc(iz->zagolovok.shirina\*iz->zagolovok.vysota\*iz-

>zagolovok.kol\_voKomponent);

// Считывание битов

fread(iz->pikseli, iz->zagolovok.shirina\*iz->zagolovok.vysota\*iz-

>zagolovok.kol\_voKomponent, 1, pFile);

// Работа с файлом завершается fclose(pFile);

// Возвращает указатель на данные изображения return iz;

}

/\*Функция записи изображения в файл .kai\*/

GLint SohranenieIzobrajeniyaVfail(const char \*szFileName)

{

FILE \*pFile; // Указатель файла

//Создание структуры для хранения изображения Izobrajenie \*iz=(Izobrajenie \*)malloc(sizeof(Izobrajenie));

GLint iViewport[4]; //Массив для хранения размеров порта просмотра

// Получение размеров порта просмотра glGetIntegerv(GL\_VIEWPORT, iViewport);

// Считывание битов из буфера цвета glPixelStorei(GL\_PACK\_ALIGNMENT, 1);

glPixelStorei(GL\_PACK\_ROW\_LENGTH, 0);

glPixelStorei(GL\_PACK\_SKIP\_ROWS, 0);

glPixelStorei(GL\_PACK\_SKIP\_PIXELS, 0);

// Переключение на передний буфер glReadBuffer(GL\_FRONT);

//Установка параметров изображения iz->zagolovok.shirina=iViewport[2];

iz->zagolovok.vysota=iViewport[3];

iz->zagolovok.formatCveta=GL\_RGB;

iz->zagolovok.formatKomponenty=GL\_UNSIGNED\_BYTE; iz->zagolovok.kol\_voKomponent=3;

//Выделение памяти для хранения битов

iz->pikseli=(unsigned char \*)malloc(iz->zagolovok.shirina\*iz->zagolovok.vysota\*iz-

>zagolovok.kol\_voKomponent);

//Чтение битов

glReadPixels(0, 0, iz->zagolovok.shirina, iz->zagolovok.vysota, iz->zagolovok.formatCveta,iz-

>zagolovok.formatKomponenty, iz->pikseli);

// Открытие файла

pFile = fopen(szFileName, "wb");

// Запись заголовка

fwrite(&(iz->zagolovok), sizeof (Zagolovok), 1, pFile);

// Запись данных об изображении

fwrite(iz->pikseli, iz->zagolovok.shirina\*iz->zagolovok.vysota\*iz-

>zagolovok.kol\_voKomponent, 1, pFile);

//Закрытие файла fclose(pFile);

// Успех return 1;

}

Izobrajenie \*izobr; //Текущее изображение

Izobrajenie \*izobr\_bmp; //Изображение из файла .bmp Izobrajenie \*izobr\_kai; //Изображение из файла .kai Izobrajenie \*izobr\_ch\_b; //Черно-белое изображение

AUX\_RGBImageRec \*pImage=NULL; //Изображение AUX\_RGB

//Переменная для хранения режима рисования static GLint rejim = 1;

// Должным образом обновляем флаги в ответ на выбор позиции из меню void ObrabotkaMenu(int punktMenu)

{

// меняем индекс режима визуализации на индекс,

// соответствующий позиции меню rejim = punktMenu;

// Активизируем перерисовывание изображения glutPostRedisplay(); } glutPostRedisplay();

}

int k;

int iz=0;

void Pererisovka(void)

{

GLint iViewport[4];

//GLbyte \*pModifiedBytes = NULL; GLfloat invertMap[256];

GLint i;

// Очищаем окно текущим цветом очистки glClear(GL\_COLOR\_BUFFER\_BIT);

// Текущее растровое положение всегда соответствует левому нижнему углу окна glRasterPos2i(0, 0);

// В зависимости от индекса режима визуализации выполняются необходимые операции с изображением

switch(rejim)

{

case 0: /\*Очистка экрана\*/ glClearColor(0.0f, 0.0f, 0.0f, 0.0f); break;

case 1: /\*Загрузка изображения из файла \*.bmp\*/ pImage=auxDIBImageLoad("C:\\photo.bmp"); izobr\_bmp=(Izobrajenie \*)malloc(sizeof(Izobrajenie)); izobr\_bmp->zagolovok.shirina=pImage->sizeX; izobr\_bmp->zagolovok.vysota=pImage->sizeY; izobr\_bmp->pikseli=pImage->data;

izobr\_bmp->zagolovok.formatCveta=GL\_RGB;

izobr\_bmp->zagolovok.formatKomponenty=GL\_UNSIGNED\_BYTE; izobr\_bmp->zagolovok.kol\_voKomponent=3;

izobr=izobr\_bmp;

break;

case 2: /\*Сохранение текущего буфера в файл\*/ glPixelStorei(GL\_PACK\_ALIGNMENT, 1); SohranenieIzobrajeniyaVfail("C:\\123.kai");

break;

case 3: /\*Загрузка изображения из собственного файла\*/ izobr\_kai=ChtenieIzobrajeniyaIzFaila("C:\\123.kai"); izobr=izobr\_kai;

break;

case 4: /\*Инверсия цветов\*/ invertMap[0] = 1.0f;

for(i = 1; i < 256; i++)

invertMap[i] = 1.0f - (1.0f / 255.0f \* (GLfloat)i); glPixelMapfv(GL\_PIXEL\_MAP\_R\_TO\_R, 255, invertMap);

glPixelMapfv(GL\_PIXEL\_MAP\_G\_TO\_G, 255, invertMap);

glPixelMapfv(GL\_PIXEL\_MAP\_B\_TO\_B, 255, invertMap); glPixelTransferi(GL\_MAP\_COLOR, GL\_TRUE);

break;

case 5: /\*Получение черно-белого изображения\*/

// Вначале рисуем изображение в буфере цвета

glDrawPixels(izobr->zagolovok.shirina, izobr->zagolovok.vysota, izobr-

>zagolovok.formatCveta, izobr->zagolovok.formatKomponenty, izobr->pikseli); // Распределяем память для карты яркости

izobr\_ch\_b=(Izobrajenie \*)malloc(sizeof(Izobrajenie)); izobr\_ch\_b->zagolovok.shirina=izobr->zagolovok.shirina; izobr\_ch\_b->zagolovok.vysota=izobr->zagolovok.vysota;

izobr\_ch\_b->zagolovok.formatCveta=GL\_LUMINANCE;

izobr\_ch\_b->zagolovok.formatKomponenty=GL\_UNSIGNED\_BYTE; izobr\_ch\_b->zagolovok.kol\_voKomponent=1;

izobr\_ch\_b->pikseli=(unsigned char \*) malloc(izobr\_ch\_b-

>zagolovok.shirina\*izobr\_ch\_b->zagolovok.vysota\*izobr\_ch\_b-

>zagolovok.kol\_voKomponent);

// Масштабируем цвета согласно стандарту NTSC glPixelTransferf(GL\_RED\_SCALE, 0.3f) ; glPixelTransferf(GL\_GREEN\_SCALE, 0.59f); glPixelTransferf(GL\_BLUE\_SCALE, 0.11f) ;

// Считываем пиксели в буфер

glReadPixels(0,0,izobr\_ch\_b->zagolovok.shirina, izobr\_ch\_b->zagolovok.vysota, izobr\_ch\_b-

>zagolovok.formatCveta,

izobr\_ch\_b->zagolovok.formatKomponenty, izobr\_ch\_b->pikseli);

izobr=izobr\_ch\_b;

// Масштабирование цвета возвращается в норму glPixelTransferf(GL\_RED\_SCALE, 1.0f) ; glPixelTransferf(GL\_GREEN\_SCALE, 1.0f); glPixelTransferf(GL\_BLUE\_SCALE, 1.0f);

break; default:

break; }

if (rejim!=0)

// Рисуются пиксели

{

glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1);

glDrawPixels(izobr->zagolovok.shirina, izobr->zagolovok.vysota, izobr-

>zagolovok.formatCveta, izobr->zagolovok.formatKomponenty, izobr->pikseli);

}

//Переключает буферы glutSwapBuffers();

}

void IzmenenieRazmera(int w, int h)

{

//Предотвращает деление на ноль, когда окно слишком маленькое if (h==0)

h=1;

glViewport(0,0,w,h);

//Система координат обновляется перед модификацией glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

}

//Точка входа основной программы int main(int argc, char\* argv[])

{

glutInit(&argc, argv); glutInitDisplayMode(GLUT\_RGB|GL\_DOUBLE); glutInitWindowSize(512,512); glutCreateWindow("Операции над пикселями"); glutReshapeFunc(IzmenenieRazmera); glutDisplayFunc(Pererisovka);

//Создается меню и добавляются опции выбора glutCreateMenu(ObrabotkaMenu);

glutAddMenuEntry("Очистить экран", 0); glutAddMenuEntry("Загрузить изображение .bmp", 1); glutAddMenuEntry("Сохранить текущее изображение", 2); glutAddMenuEntry("Загрузить изображение из файла .kai", 3); glutAddMenuEntry("Инверсия цветов", 4);

glutAddMenuEntry("Черно-белое изображение", 5); glutAttachMenu(GLUT\_RIGHT\_BUTTON); gluOrtho2D(0,512,0,512);

glClearColor(0.0f, 0.0f, 0.0f, 0.0f); rejim=0;

glutMainLoop();

// Освобождаем исходные данные изображений free(izobr);

free(izobr\_ch\_b); free(izobr\_kai); free(izobr\_bmp);

free(pImage);

return 0;

}