Отрисовка примитивов в OpenGl. 2D Текстурированние.

Среда работы – IDE Code::Blocks.

Цель работы: получения представления о работе с координатной системой, отрисовкой примитивов, их преобразованием (аффинные преобразования) и 2D текстурирование.

Задание: написать код заготовку элементов интерфейса, для отладки работы спрайт листа.

Создание базового проекта OpenGL в CB

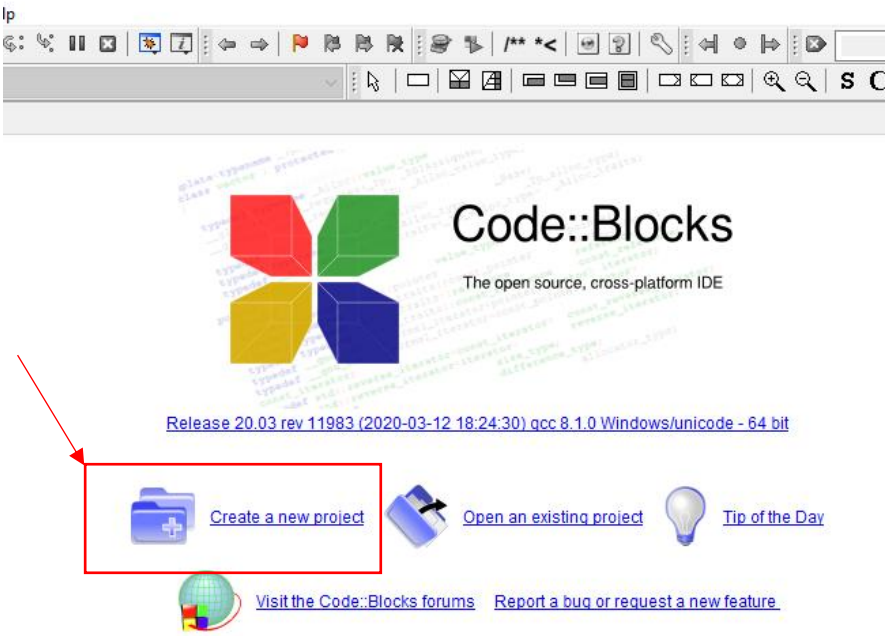

Рис. 1, Приветственное окно CB

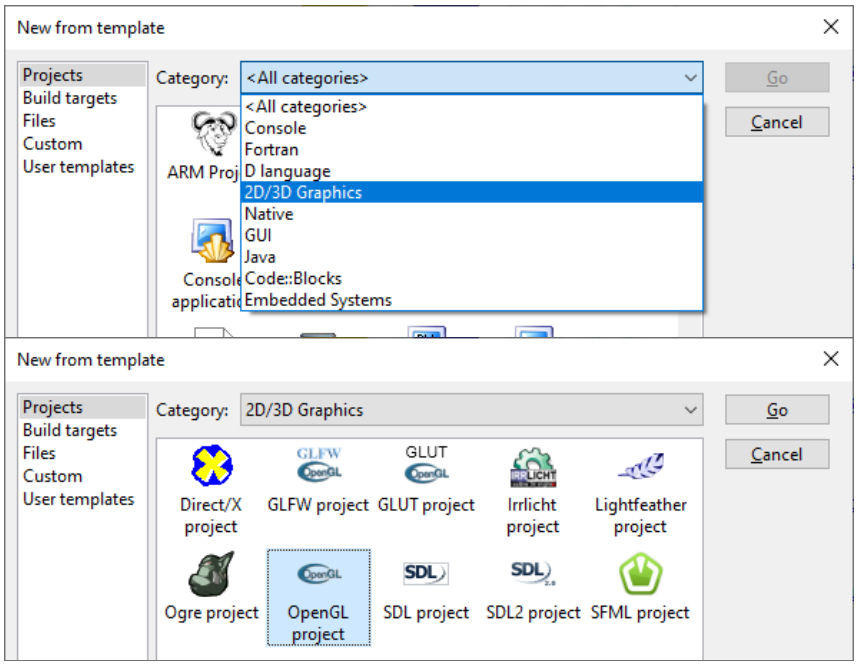

Рис. 2, Создание проекта с шаблоном OpenGL

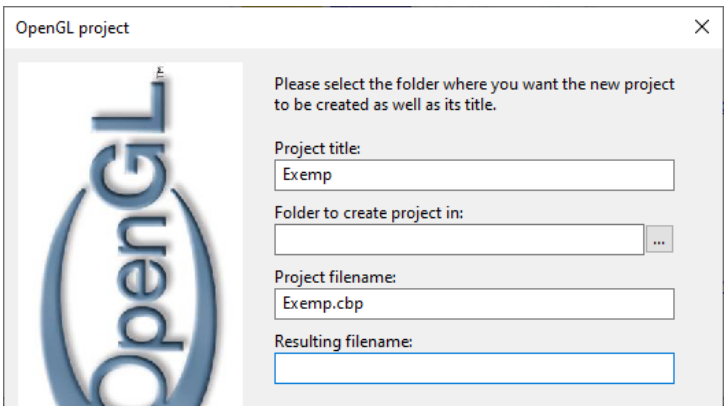

Рис. 3, Название проекта и путь до папки сохранения

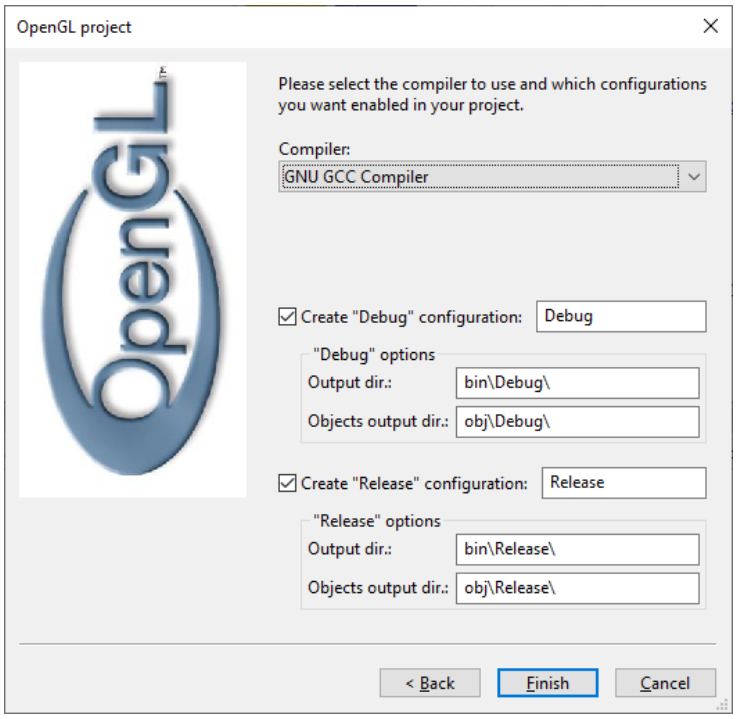

Рис. 4, Проверка на нужный компилятор и отметки

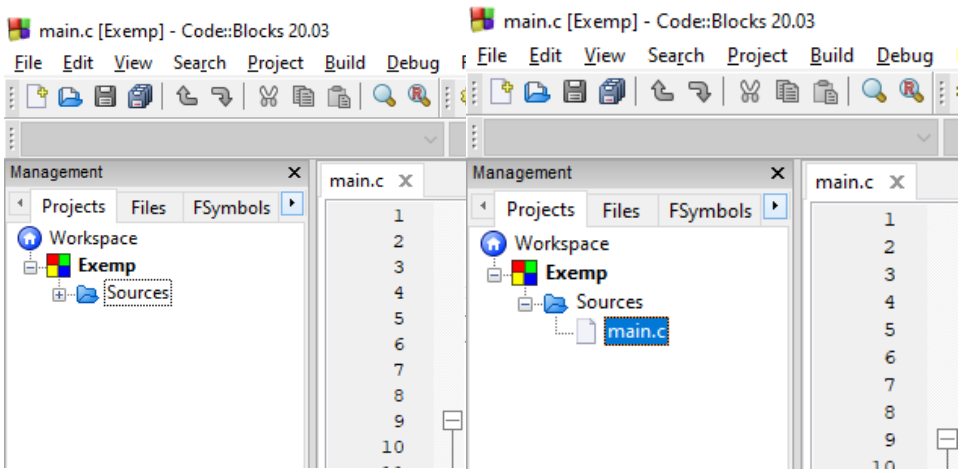

Рис. 5, Расположение основного файла

main.c

```
59      /* program main loop */
60      while (!bQuit)
61      {
62          /* check for messages */
63          if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
64          {
65              /* handle or dispatch messages */
66              if (msg.message == WM_QUIT)
67              {
68                  bQuit = TRUE;
69              }
70              else
71              {
72                  TranslateMessage(&msg);
73                  DispatchMessage(&msg);
74              }
75          }
76          else
77          {
78              /* OpenGL animation code goes here */
79
80              glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
81              glClear(GL_COLOR_BUFFER_BIT);
82
83              glPushMatrix();
84              glRotatef(theta, 0.0f, 0.0f, 1.0f);
85
86              glBegin(GL_TRIANGLES);
87
88                  glColor3f(1.0f, 0.0f, 0.0f);   glVertex2f(0.0f,   1.0f);
89                  glColor3f(0.0f, 1.0f, 0.0f);   glVertex2f(0.87f,  -0.5f);
90                  glColor3f(0.0f, 0.0f, 1.0f);   glVertex2f(-0.87f, -0.5f);
91
92              glEnd();
93
94              glPopMatrix();
95
96              SwapBuffers(hDC);
97
98              theta += 1.0f;
99              Sleep (1);
100         }
101     }
```

Рис. 6, Основной цикл кода

Подключение библиотек

Для работы могут потребоваться библиотеки, и подключение сторонних файлов в исполняемом проекте CB делается через добавления файла.

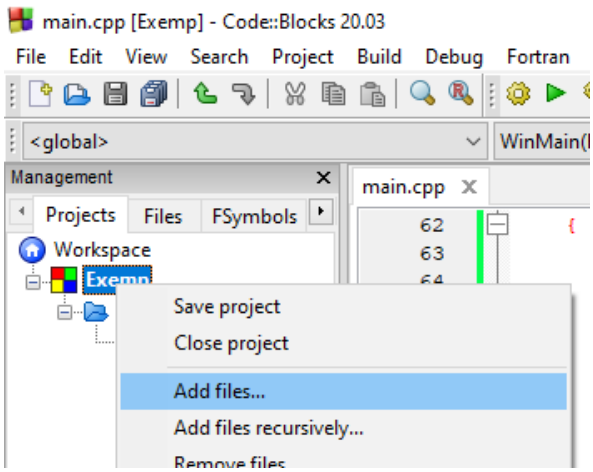

Рис. 7, Подключение любого существующего файла

Библиотека stb сделана для упрощения загрузки изображений и вывода текста, как с текстового шаблона в виде изображения или набором примитивов.

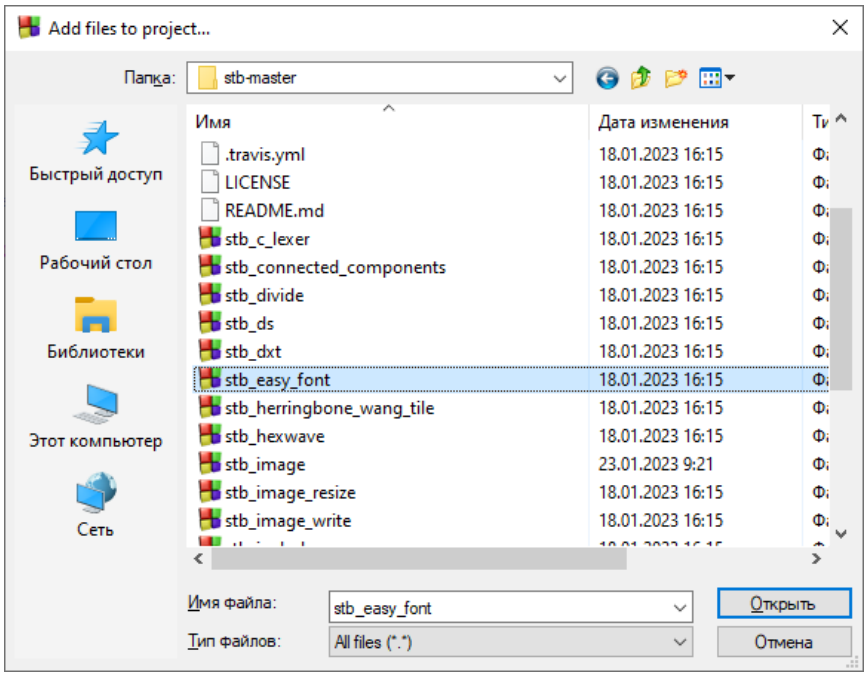

Рис. 8, Выбираем stb_easy_font и stb_image

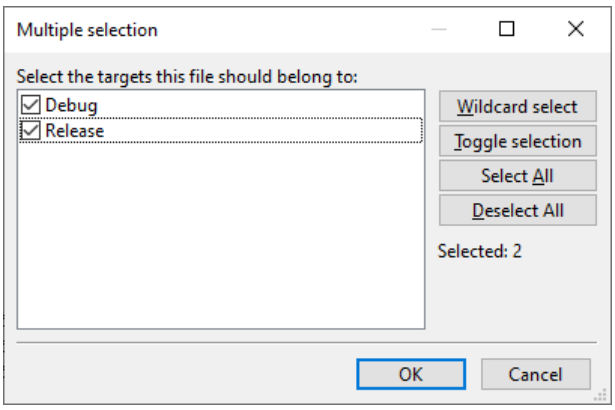

Рис. 9, Добавляем привязку при отладке и релизе

Обратите внимание – сторонняя библиотека подключается в “…”, вместо <…> для стандартных библиотек.

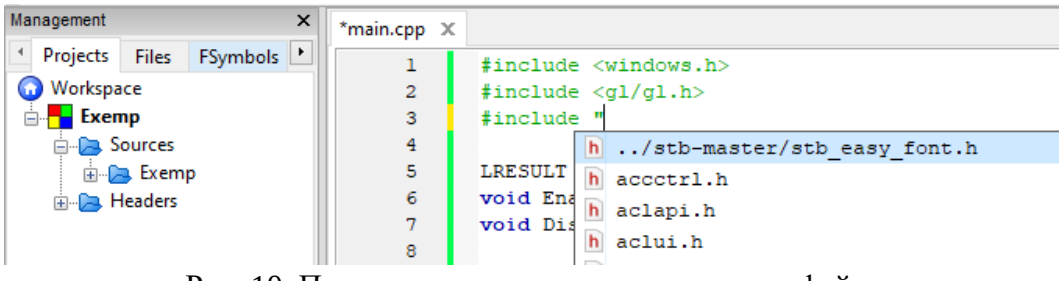

Рис. 10, Подключение внутри исполняемого файла

Настройка координат окна вывода

Изначально координаты экрана нормализированы и расположены как на Рис. 11.

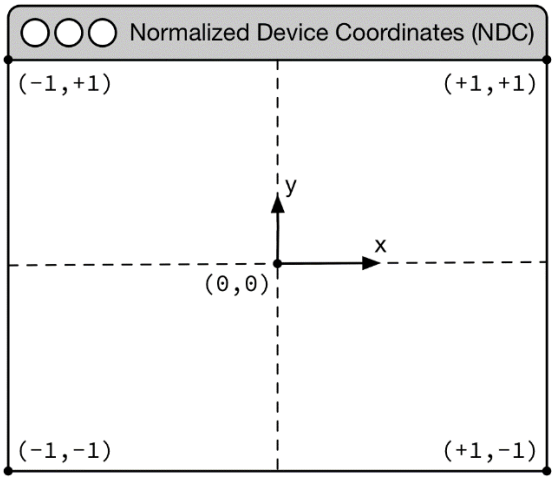

Рис. 11 Нормализированные координаты

Для изменения и приведения координат экрана существуют 2 основные функции glOrtho и glFrustrum.

glOrtho описывает матрицу перспективы, которая создает параллельную проекцию. Параметры (слева, внизу, рядом) и (справа, сверху, рядом) указывают точки на плоскости обрезки, сопоставленные с левыми и правыми нижними углами окна соответственно, при условии, что глаз находится в точке (0, 0, 0). Дальний параметр указывает расположение плоскости вырезки. Как zNear, так и zFar могут быть положительными или отрицательными. Соответствующая матрица показана на следующем рисунке.

glFrustum описывает матрицу перспективы, которая создает проекцию перспективы. Параметры (слева, внизу, zNear) и (справа, сверху, zNear) указывают точки на близком плоскости вырезки, сопоставленные с левыми и верхними правыми углами окна соответственно, при условии, что глаз находится в (0,0,0). Параметр zFar указывает расположение дальней плоскости вырезки. Как zNear, так и zFar должны быть положительными.

Для приведения координат в более понятную используйте конструкцию:

```
…
RECT rct; //создание переменной с координатами прямоуголника
   GetClientRect(hwnd, &rct); //получение текущих координат окна
   glOrtho(0, rct.right, 0, rct.bottom, 1, -1); //выставляем их как координаты окна
…
```

после мы получаем координаты как на Рис.12.

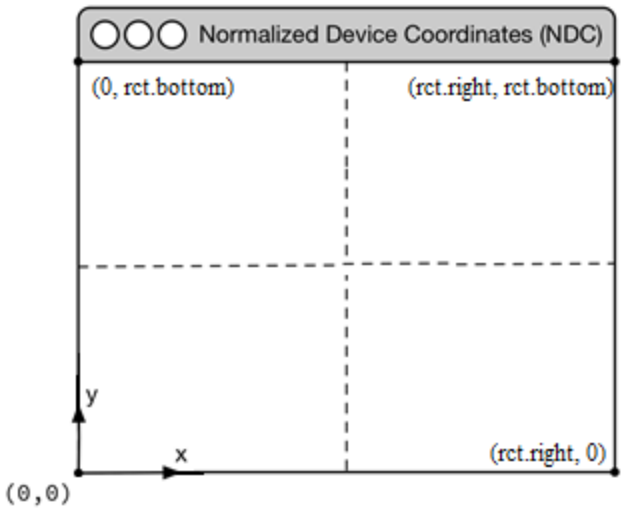

Рис. 12 Полученная система координат
(не забывайте полученные координаты инвертированы по вертикали)

```
/* create main window */
hwnd = CreateWindowEx(0,
                      "GLSample",
                      "OpenGL Sample",
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      256,
                      256,
                      NULL,
                      NULL,
                      hInstance,
                      NULL);
```

Рис. 13 Размер окна по X и Y в px

Отрисовка примитивов в OpenGl

Все геометрические примитивы в OpenGL задаются вершинами (набором чисел, определяющих их координаты в пространстве).

OpenGL работает с однородными координатами (x, y, z, w). Если координата z не задана, то она считается равной нулю. Если координата w не задана, то она считается равной единице.

Под линией в OpenGL подразумевается отрезок, заданный своими начальной и конечной вершинами.

Под гранью (многоугольником) в OpenGL подразумевается замкнутый выпуклый многоугольник с несамопересекающейся границей.

Все геометрические объекты в OpenGL задаются посредством вершин, а сами вершины задаются процедурой:

glVertex{2 3 4}{s i f d}[v](TYPE x, …),

где реальное количество параметров определяется первым суффиксом (2, 3 или 4), а суффикс v означает, что в качестве единственного аргумента выступает массив, содержащий необходимое количество координат. Например:

```
glVertex2s(1, 2);
glVertex3f(2.3, 1.5, 0.2);
GLdouble vect[] = {1.0, 2.0, 3.0, 4.0};
glVertex4dv(vect);
```

Для задания геометрических примитивов необходимо как-то выделить набор вершин, определяющих этот объект. Для этого служат процедуры glBegin() и glEnd(). Процедура glBegin(GLenum mode) обозначает начало списка вершин, описывающих геометрический примитив. Тип примитива задаётся параметром mode, который принимает одно из следующих значений:

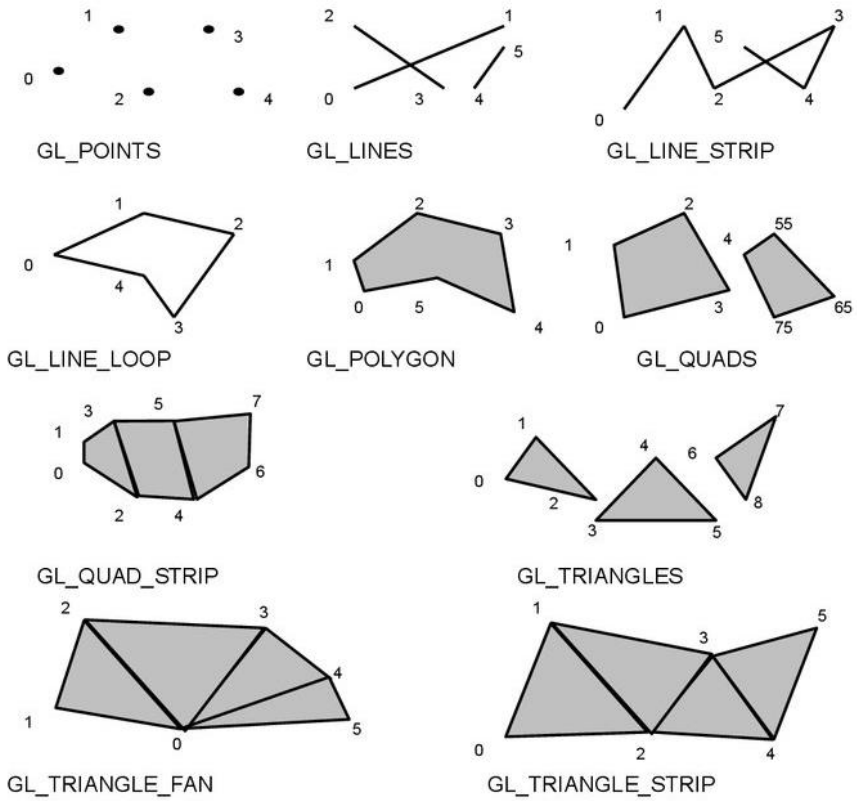

Рис. 14 Основные примитивы OpenGl

Отрисовка через VBO и VAO

Vertex Buffer Object (VBO) — это средство OpenGL, позволяющее загружать определенные векторы данных в память GPU. Например, если вы хотите сообщить GPU координаты вершин, цвета или нормали, можно создать VBO и записать эти данные в него, а потом пересылать указатели на эти вершины для их построения или редактирования. При работе с большим количеством вершин это более оптимальный способ.

```
        …
Button btn1 = Button[buttonId]; // передача элемента типа TBtn в буфер
glVertexPointer(2, GL_FLOAT, 0, btn1.vert); // подготовка к использованию VBO и
                                                    указание нужного буфера
   glEnableClientState(GL_VERTEX_ARRAY); //разрешение на использование
   if(btn1.isDown)glColor3f(0.2,1,0.2);  // различные цвета в зависимости от нажатия
   else if (btn1.isHover)glColor3f(0.8,0.8,1);
   else glColor3f(0.6,0.6,0.6);

   glDrawArrays(GL_TRIANGLE_FAN, 0, 4); // отрисовка примитива

   glColor3f(0.5,0.5,0.5);
   glLineWidth(1);
   glDrawArrays(GL_LINE_LOOP,0,4); // отрисовка линии по тем же координатам для
                                                        рамки примитива
   glDisableClientState(GL_VERTEX_ARRAY); // блокировка использования VBO
        …
```

Вывод текста с примитивом

Как наиболее простой вариант вывода текста в stb_image идет через набор квадратов. В библиотеке для каждой из букв латинского алфавита занесен набор квадратов относительно друг друга обрисовывающие буквы.

Для удобства можно выделить в отдельную структуру данные о кнопке:

```
typedef struct
{
   char name[nameLen]; // длина имени кнопки
   float vert[8]; // 4 вершины по 2 координаты
   //text
   float buffer[50*nameLen]; // для координат примитивов
   int num_quads; //для количества
   float textPosX,textPosY,textS; // координаты расположения текста и
  //масштабный коэфициент
} Button;
```

Пример функции добавления кнопки:

```
int AddButton(char *name, float x, float y, float width, float height, float textS)
{
   btnCnt++; //заводим счетчик кнопок
   btn = (Button*)realloc(Button, sizeof(Button [0])*btnCnt); //выделяем память под нужное
                                                        //количество
   snprintf(Button[btnCnt-1].name, nameLen, "%s", name); //выделение памяти и запись
                                                    //имени
   float *vert = btn[btnCnt-1].vert; //передача координат кнопки
   vert[0]=vert[6]=x;
   vert[2]=vert[4]=x+width;
   vert[1]=vert[3]=y;
   vert[5]=vert[7]=y+height;
```

```
    Button *b= btn + btnCnt - 1; //записываем в буфер данные кнопки
    b->num_quads = stb_easy_font_print(0, 0, name, 0, b->buffer, sizeof(b->buffer)); // запись
                                                        //координат вершин элементов имени
    b->textPosX = x +(width-stb_easy_font_width(name)*textS)/2.0;
    b->textPosY = y +(height-stb_easy_font_height(name)*textS)/2.0;
    b->textPosY+= textS*2;
    b->textS = textS;
    return btnCnt-1;
}
```

После все это нужно отрисовать кнопку

```
void ShowButton(int buttonId)
{

    Button btn1 = btn[buttonId]; //пересылка всех данных в буфер
    glVertexPointer(2, GL_FLOAT, 0, btn1.vert); //запись координат в VBO
    glEnableClientState(GL_VERTEX_ARRAY); //разрешение
    glColor3f(0.2,1,0.2); //цвет кнопки

    glDrawArrays(GL_TRIANGLE_FAN, 0, 4); //отрисовка кнопки

    glColor3f(0.5,0.5,0.5); //цвет обводки
    glLineWidth(1); // толщина обводки кнопки
    glDrawArrays(GL_LINE_LOOP,0,4); //отрисовка обводки
    glDisableClientState(GL_VERTEX_ARRAY);

    glPushMatrix(); //матрицу в стек
    glColor3f(0.5,0.5,0.5); //цвет текста
    glTranslatef(btn1.textPosX,btn1.textPosY,0); //перенос матрицы для отрисовки текста
    glScalef(btn1.textS,btn1.textS,0); //масштабирование текста
    glEnableClientState(GL_VERTEX_ARRAY); // разрешение
    glVertexPointer(2, GL_FLOAT, 16, btn1.buffer);  //вектор для отрисовки
    glDrawArrays(GL_QUADS, 0, btn1.num_quads*4); //отрисовка текста
    glDisableClientState(GL_VERTEX_ARRAY);
    glPopMatrix();
}
```

Для создания уникальных шрифтов можно использовать как такой же набор букв в виде набора примитивов, писать свой векторный вариант для каждого используемого символа или создание своей текстуры со всеми символами.

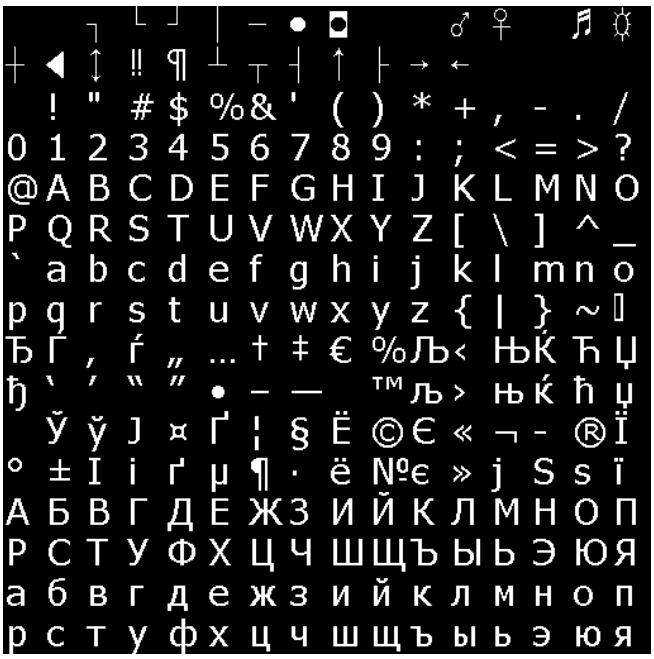

Рис. 15 Пример символьной текстуры

Текстурированние

Текстурирование позволяет наложить изображение на многоугольник и вывести этот многоугольник с наложенной на него текстурой, соответствующим образом преобразованной. OpenGL поддерживает одно- и двумерные текстуры, а также различные способы наложения текстуры.

Для использования текстуры надо сначала разрешить одно- или двумерное текстурирование при помощи команд glEnable(GL_TEXTURE1D) или glEnable(GL_TEXTURE_2D).

Для задания двумерной текстуры служит процедура:

glTexlmage2D(GLenum target, GLint level, GLint component,

GLsizei width, GLsizei height, GLint border, GLenum format,

GLenum type, const GLvoid *pixels).

Параметр target зарезервирован для будущего использования и в текущей 17 версии OpenGL должен быть равен GL_TEXTURE_2D. Параметр level используется в том случае, если задается несколько разрешений данной текстуры. При ровно одном разрешении он должен быть равным нулю.

Следующий параметр – component – целое число от 1 до 4, показывающее, какие из RGBA-компонентов выбраны для использования. Значение 1 выбирает компонент R, значение 2 выбирает R и A компоненты, 3 соответствует R, G и B, а 4 соответствует компонентам RGBA.

Параметры width и height задают размеры текстуры, border задает размер границы (бортика), обычно равный нулю. Как параметр width, так и параметр height, должны иметь вид 2n + 2b, где n – целое число, а b –

значение параметра border. Максимальный размер текстуры зависит от реализации OpenGL, но он не менее 64 на 64.

Способ выбора соответствующего текселя как для увеличения, так и для уменьшения (сжатия) текстуры необходимо задать отдельно. Для этого используется процедура: glTexParameteri(GL_TEXTURE_2D, GLenum p1, GLenum p2), где параметр p1 показывает, задается ли фильтр для сжатия или для растяжения текстуры, принимая значение GL_TEXTURE_MIN_FLITER или GL_TEXTURE_MAG_FILTER. Параметр p2 задает способ фильтрования.

Для правильного применения текстуры каждой вершине следует задать соответствующие ей координаты текстуры при помощи процедуры:

glTexCoord{1 2 3 4}{s i f d}[v](TYPE coord, ...).

Этот вызов задаёт значения индексов текстуры для последующей команды glVertex. Если размер грани больше, чем размер текстуры, то для циклического повторения текстуры служат команды:
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_S_WRAP, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_T_WRAP, GL_REPEAT).

Для загрузки текстуры из файлов изображения используется дополнительная h. файл из библиотека stb_image с одноименным названием. В заголовочном файле приведен пример использования.

```
…
// Basic usage (see HDR discussion below for HDR usage):
//    int x,y,n;
//    unsigned char *data = stbi_load(filename, &x, &y, &n, 0);
//    // ... process data if not NULL ...
//    // ... x = width, y = height, n = # 8-bit components per pixel ...
//    // ... replace '0' with '1'..'4' to force that many components per pixel
//    // ... but 'n' will always be the number that it would have been if you said 0
//    stbi_image_free(data)
…
```

И как вариант реализации загрузки текстуры:

```
   int twidth, thight, tcnt; //переменные ширины, высоты,
   unsigned char *data=stbi_load(filename,&twidth,&thight,&tcnt,0);  // в поле filename,
        //прописывается имя файла “image.png”, а файл хранится в директории проекта
glGenTextures(1, textureID); //генерация текстуры
   glBindTexture(GL_TEXTURE_2D, *textureID);
   glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,swarp); //настройки
   glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,twarp);
   glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,filter);
   glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,filter);
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, twidth, thight,
            0, tcnt == 4 ? GL_RGBA : GL_RGB, GL_UNSIGNED_BYTE, data);
   glBindTexture(GL_TEXTURE_2D, 0); //задание текстуры
```

```
stbi_image_free(data); //освобождение буффера
```

Отрисовка текстуры

Для отображения необходимы координаты текстурируемого объекта и текстурные координаты нашего изображения

```
…
static float svertix[]= {0,0, 1,0, 1,1, 0,1}; //вектор текстурируемого многоугольника
static float TexCord[]= {0,0, 1,0, 1,1, 0,1}; // текстурные координаты изображения

glEnable(GL_TEXTURE_2D); //разрешение использования текстуры
glBindTexture(GL_TEXTURE_2D, texture);
glEnable(GL_ALPHA_TEST); // проверка на элементы α-канала (не обязательно)
glAlphaFunc(GL_GREATER, 0.99); // задается типе уровня и его числовая граница
glEnableClientState(GL_VERTEX_ARRAY); //использование вектора координат
glEnableClientState(GL_TEXTURE_COORD_ARRAY); //использование вектора
                                                    // текстурных координат
glVertexPointer(2, GL_FLOAT, 0, svertix); //используем вектор координат
glTexCoordPointer(2, GL_FLOAT, 0, TexCord); //используем вектор текстурных координат
static float spriteXsize=800; //переменные с размерами текстуры и отдельного кадра
static float spriteYsize=80;
static float charsizey=80;
static float charsizex=100;
vrtcoord.left=(charsizex*n)/spriteXsize; //вычисление координат кадра на изображении от
vrtcoord.right=vrtcoord.left+(charsizex/spriteXsize);                    //номера кадра
vrtcoord.top=(charsizey*t)/spriteYsize;
vrtcoord.bottom=vrtcoord.top+(charsizey/spriteYsize);

TexCord[1] = TexCord[3] = vrtcoord.bottom; // запись в вектор текстурных координат
TexCord[5] = TexCord[7] = vrtcoord.top;
TexCord[2] = TexCord[4] = vrtcoord.left;
TexCord[0] = TexCord[6] = vrtcoord.right;

glDrawArrays(GL_TRIANGLE_FAN,0,4); //отрисовка текстурированного объекта

glDisableClientState(GL_VERTEX_ARRAY); //отключение работы с вектором
glDisableClientState(GL_TEXTURE_COORD_ARRAY);
glDisable(GL_ALPHA_TEST); //отключение проверки α-канала
…
```

Обработка сигналов периферии

Посоле главного цикла имеется функция, принимающая сигналы периферии в окно при нахождении его в фокусе:

```
…
LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam,
LPARAM lParam)
…
```

где отслеживается нахождение окна в фокусе и через switch – case принимается сигнал с периферии.

В каждый case записывается свой сигнал и функции которые будут выполнены при его получении.

Пример:

```
…
switch (uMsg)
   {
   case WM_CLOSE: //закрытие при нажатии на закрытие окна
      PostQuitMessage(0); //закрытие окна
      break;

   case WM_DESTROY:
      return 0;

   case WM_MOUSEMOVE: //при событии - перемещении мыши
     Menu_MouseMove(LOWORD(lParam), HIWORD(lParam));//пересылка координат мыши
      break;                                                    // в функцию
…
```