ЛАБОРАТОРНАЯ РАБОТА №3
2D Текстурирование в OpenGL. Простое перемещение.

Цель работы: сделать простое 2D текстурирование и перемещение.

Задание: написать код для перемещения кадра спрайт листа, сделать обработку перемещения.

Среда работы: IDE Code::Blocks.

Текстурирование

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.
- ✓ 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.
- ✓ Параметр 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 x 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 библиотека stb_image. 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)
…
```

Реализация 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 буфера
```

Отображение 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 тип 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 номера
vrtcoord.right=vrtcoord.left+(charsizex/spriteXsize);                  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

После 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 в функцию
      break;
        …
```

Реализация перемещения

Для перемещения объекта необходимо каждый кадр просчитывать координаты места отрисовки текстурированного примитива или объекта. Для этого необходимы координаты нашего объекта, скорость изменения координаты.

Пример реализации:

```
…
float speed = 2.5;                    //скорость изменения координат
float gravity = 0.2;                  // скорость изменения координаты Y
                                      //(для реализации падения)
typedef struct
{
   float x, y, dx, dy;                //обе координаты и скорость изменения
} Hero;
…
```

Для более детального описания физическое поведение объекта, необходимо добавить еще переменные описывающие ускорение объекта (изменение dx, dy).

Для отрисовывания объекта в определенном месте необходимо провести инициализацию:

```
void Hero_Init(Hero *obj, float x1, float y1, float dx1, float dy1)
{
   obj->x=x1;
   obj->y=y1;
   obj->dx=dx1;
   obj->dy=dy1;
}
```

Функция обработки перемещения:

```
void Hero_Move(Hero *obj)
{
   obj->y+=obj->dy;                          //обновление координат
   obj->x+=obj->dx;
   obj->dy-=gravity;                         // влияние примера гравитации
   if (GetKeyState(VK_LEFT)<0 && State==1)   // проверка состояния
                                             //программы и нажатия клавиши влево
   {
currentframe+=0.15;                          // подсчет для смены кадра по спрайт
```

```
                                        //листу (не оптимально)
obj->dx-=speed;                          // пересылка изменения координаты
obj->x+=obj->dx;                         //вычисление конечной координаты
obj->dx=0;                               // обнуление для исключение пересчета
                                         // при обновлении координаты в начале
if (currentframe>8) currentframe-=7;     //обнуление номера кадра для
                                        //цикличности
      //cout << currentframe <<';'<< directional <<';' << obj->x << ';' << obj->y <<
endl;
//фрагмент для проверки текущей координаты
   }
   if (GetKeyState(VK_RIGHT)<0 && State==1) // повторение предыдущего
условия
   {
      currentframe+=0.15;
      obj->dx+=speed;
      obj->x+=obj->dx;
      obj->dx=0;
      if (currentframe>8) currentframe-=7;
      directional=0;
      //cout << currentframe <<';'<< directional <<';' << obj->x << ';' << obj->y <<
endl;

   }
   if (GetKeyState(VK_UP)<0 && (obj->y<20) && State==1)
                                         //повторение, с условием проверки
                                         // нахождения на определенном уровне
   {
      obj->dy =speed*1.2;                // дает импульс для имитации прыжка
      obj->y+=obj->dy;                   //расчет координаты Y
   }
}
```

При данной реализации перемещения, применяя в сочетании с изменением системы координат glOrtho, изображение просто упадет за край видимой области. Для ограничения необходимо добавить простое ограничение:

…

```
void Reflect (float *da, float *a, BOOL cond, float wall)
{
   if (!cond) return;                                  //условие проверки пересечения
                                                       //координаты и препятствия
   *da*=-0,1;                                          // пересчет изменения координаты
                                                       //в начале функции перемещения
   *a=wall;                                           //перемещение в координату стены
}
…
```

Пример записи:

```
…
Reflect(&obj->dy, &obj->y, obj->y<0,0);
…
```