Лабораторная работа №3 «Простейшие обработчики событий»

Для обработки событий клавиш необходимо создать еще один обработчик событий **onKeyDown**. В простейшем случае тело обработчика **onKeyDown** может состоять и одной команды Refresh, которая вызывает обработчик перерисовки окна.

Например, если в обработчике **onPaint** нарисовать многоугольник, состоящий из связных треугольников, цвет которых задается случайным образом

```
Randomize;
glBegin(GL_TRIANGLE_FAN);
glVertex2f (0, 0); // вершина, общая для всех треугольников
For i:= 0 to 6 do begin
glColor3f(random, random, random);
glVertex2f(0.5*(2*Pi*6), 0.5*sin (2*Pi*6));
end;
```

то обработчик **onKeyDown** вида

```
if key=VK_SPACE then Refresh;
```

будет вызывать каждый раз по нажатию кнопки «пробел» перерисовку окна, и соответственно смену цвета треугольников на экране.

Key — это параметр процедуры **onKeyDown**, который представляет собой код нажатой клавиши. Для клавиш влево, вправо и т.д. необходимо проводить сравнение вида if key=VK_LEFT, if key=VK_RIGHT и т.д. Для клавиш от до 9 будут сравнения вида if Key=49 (для 1), if key=50 (для 2) и т.д. Т.е. буквенно-цифровые клавиши определяются по коду символа.

Для обработки событий мыши используется обработчик **onMouseMove**, если необходимо отследить позицию курсора. В этот метод в качестве параметра передаются X и Y, которые соответствуют текущей позиции курсора на экране. Объявим в разделе private переменные

```
xpos: GLfloat; // координаты курсора в системе координат OpenGL
ypos: GLfloat;
```

Они будут задавать координаты вершин, используемых при построении сцены. Код обработчика **onMouseMove**

```
xpos := 2*X/ClientWidth-1;
ypos := 2*(ClientHeight-Y)/ClientHeight-1;
Refresh; // перерисовываем окно
```

Здесь определяются значения переменных **xpos** и **ypos** в зависимости от текущей позиции курсора. Затем эти значения будут использованы в **onPaint** для задания координат вершин.

В обработчик **onPaint** добавить следующий код:

```
For i:= 1 to 40 do begin
  glColor3f(random, random, random); // случайного цвета
  glPointSize(random (10)); // случайного размера
  glBegin(GL_POINTS); // со случайными координатами вокруг курсора
  glVertex2f(xpos+0.2*random*sin(random(360)), ypos+0.2*random*cos(random(360)));
  glEnd;
end;
```

Еще один пример с отслеживанием позиции курсора

```
glEnable (GL_LINES_TRIPPLE);
For i:= 1 to 100 do Begin
  glRotatef(-45, 0.0, 0.0, 1.0);
  glBegin(GL_POLYGON);
  glVertex2f(0.1, -0.1);
  glVertex2f(0.1, 0.4);
  glVertex2f(0.6, 0.4);
  glVertex2f(0.6, -0.1);
  glEnd;
End;
```

Все объекты в OpenGL рисуются в точке отсчета системы координат, а команда **glRotate** осуществляет поворот системы координат.

Когда на экране присутствует несколько объектов, повернутых относительно друг друга перед рисованием очередного объекта необходимо осуществлять поворот, а после рисования – возвращать точку зрения или осуществлять следующий поворот с учетом текущего положения точки зрения.

Перенос точки зрения (системы координат) осуществляется командой **glTranslatef(x, y, z)** с тремя аргументами – величинами переноса для каждой из осей. Все сказанное по поводу восстановления точки зрения справедливо и в отношении переноса.

Для осуществления переноса по нажатию клавиш необходимо выполнить следующие действия:

- в секции private объявить переменную x.
- в обработчике **onkeyDown** написать

```
If key=VK_LEFT then begin
  x:=x-0.1;
  InvalidateRect(Handle, nil, False);
end,
if key=VK_RIGHT then begin
  x:=x-0.1;
  InvalidateRect(Handle, nil, False);
end;
```

Команда **InvalidateRect** является аналогом команды **Refresh**, но работает более качественно, поскольку основана на «низкоуровневых» методах.

- в обработчик **onPaint** добавить код

```
glTranslatef(x, 0, 0);
glBegin(GL_QUADS);
  glVertex2f(0.5, 0.5);
  glVertex2f(0.5, -0.5);
  glVertex2f(-0.5, -0.5);
  glVertex2f(-0.5, 0.5);
glEnd;
glTranslatef(-x, 0, 0);
```

Такая команда **glTranslatef** позволяет осуществлять перенос только по оси x, по осям y и z отсутствие переноса задается 0. В данном примере важен обратный перенос, поскольку в противном случае каждая новая команда переноса будет работать с уже измененной системой координат и результат движения будет отличаться от ожидаемого (можно попробовать убрать обратный перенос и посмотреть).

Задание: осуществлять вращение примитива и перемещение по всем направлениям с помощью нажатия клавиш.

Если команды поворота и переноса используются совместно, то важен их порядок: сначала должна идти команда переноса, а затем поворота, поскольку фактически при повороте и переносе мы работаем с видовой матрицей, а для матриц важен порядок при умножении.

Следующий пример позволяет, используя разложение по окружности разместить с помощью команд поворота и переноса квадраты на сцене

```
for i:=0 to 6 do begin
  glTranslatef(-0.7*cos(Pi*i/3), 0.7*sin(Pi*i/3), 0.0);
  glRotatef(-60*i, 0, 0, 1);
  glBegin(GL_QUADS);
    glVertex2f(0.25, 0.25);
    glVertex2f(0.25, -0.25);
    glVertex2f(-0.25, -0.25);
    glVertex2f(-0.25, 0.25);
  glEnd;
  glRotatef(60*i, 0, 0, 1);
  glTranslatef(0.7*(Pi*i/3), -0.7*sin (Pi*i/3), 0.0);
end;
```

При возврате к первоначальному состоянию системы координат команды поворота и переноса задаются в обратном порядке.

Задание: организовать вращение всех фигур на сцене по нажатию клавиш по и против часовой стрелки.