

# GLEngine2D

Движок основан на OpenGL.

Ориентирован на то что бы заменить собой функции рисования TCanvas.

Возможности:

- цвет в формате RGBA (альфаканал)
- разные возможные варианты работы с альфаканалом (смешивание, добавление)
- точки
- линии
- градиентные линии
- кривая по типу молнии
- стрелочки
- треугольники
- градиентные треугольники
- четырёхугольники
- прямоугольники
- градиентные четырёхугольники
- эллипс под углом
- текст
- отрисовка изображений TGA BMP JPEG PNG GIF (с альфаканалом и смешиванием цветов)
- поддержка анимации
- аппаратное сглаживание (AntiAlias)
- рисование в текстуру (FBO)
- экспорт текстуры в файл или в TBitMap
- аппаратное ускорение
- очень просто

# Оглавление

GLEngine2D.....	1
1. Инициализация.....	3
2. Подготовка к рисованию и дополнительные возможности.....	4
2.1 Работа с цветами и режимами рисования.....	4
3. Графические примитивы.....	6
3.1 Точки.....	6
3.2 Линии.....	6
3.3 Треугольники .....	6
3.4 Четырёхугольники и прямоугольники .....	6
3.5 Круги, окружности и эллипсы .....	7
3.6 Текст.....	7
3.7 Прочие примитивы.....	8
4. Работа с изображениями.....	9
4.1 Загрузка и создание.....	9
4.2 Отрисовка изображений.....	9
4.3 Рисование на изображении (FBO).....	9
4.4 Сохранение.....	10
5. Работа с шейдерами.....	11
5.1 Введение (взято с gamedev.ru и elite-games.ru).....	11
5.2 Загрузка и создание.....	11
5.3 Использование.....	11

# 1. Введение

Много уже написано, относительно использования графики в Delphi, класс Tcanvas – знаком если не всем, то многим точно. И все прекрасно знают его преимущества и недостатки. Класс конечно хорош, продуман, но с тех пор когда он создавался минуло много лет... Прогресс не стоит на месте и почти в каждом более-менее современном ПК стоит более-менее современная видеокарта.

# 1. Инициализация

Перед тем как приступить к рисованию, необходимо создать экземпляр класса TGLEngine и произвести инициализацию движка.

**Procedure** TGLEngine.VisualInit(dc: HDC; width, height: word; AntiAlias: integer);

dc – поверхность рисования

width, height — ширина и высота холста

AntiAlias — сглаживание (0 — выкл)

## Пример

```
Uses
  GLEngine // Добавляем юнит движка
...
Var
  GLE:TGLEngine; // экземпляр класса
  im:Cardinal;
....
GLE:=TGLEngine.Create; // создаём движок
// Указываем на чём рисуем и каким размером; 2 - 2х сглаживание
GLE.VisualInit(GetDC(Panell1.Handle), Panell1.ClientWidth, Panell1.ClientHeight, 2);
```

## 2. Подготовка к рисованию и дополнительные возможности

Рассмотрим функции которые обеспечивают общие настройки движка и самого процесса рисования. Начало рисования должно начинаться со строк

```
GLE.BeginRender;
```

а заканчиваться строками

```
GLE.FinishRender;
```

Только после вызова FinishRender – будет осуществлён вывод сцены на поверхность рисования.

### Пример

```
Gle.BeginRender;  
Gle.SetColor(1,1,1,1);  
Gle.DrawImage(0,0,640,480,0,false,pod);  
Gle.DrawImage(0,0,640,480,0,false,CreateTex);  
Gle.FinishRender;
```

Для изменения размера холста используется метод:

```
Procedure Resize(w,h:integer);
```

w,h – новые значение ширины и высоты холста соответственно.

Полезными будут следующие методы:

```
function GetTimeDrawFrame:double;  
function GetFPS:Integer;
```

GetTimeDrawFrame – возвращает время (в миллисекундах) затраченное на прорисовку последнего кадра. GetFPS – количество кадров реально нарисованных за последнюю секунду.

```
Function GetGLEngineWidth:word;  
Function GetGLEngineHeight:word;
```

GetGLEngineWidth, GetGLEngineHeight – возвращают ширину и высоту области рисования, указанной при инициализации или изменении размеров этой области.

```
Procedure Clear;
```

Clear — очищает область вывода.

### 2.1 Работа с цветами и режимами рисования

Для установки цвета выводимой фигуры (изображения) используется метод

```
Procedure SetColor(r,g,b,a:single);Overload;  
Procedure SetColor(color:TGLColor);Overload;
```

где TGLColor это

```
TGLColor = record  
  Red, Green, Blue, alpha:single;  
end;
```

Для быстрого получения цвета в TGLColor можно использовать метод:

```
function ColorGL(r,g,b,a:single):TGLColor;
```

Для установки цвета фона:

```
Procedure SetBKColor(r,g,b:single);
```

В движке предусмотрен вывод линий со сглаживанием. Включить или выключить его можно методом:

```
Procedure AntiAlias(Enable:boolean);
```

Обратите внимание, что значение величины сглаживания — задаётся при инициализации и потом его изменить нельзя. Если при инициализации выставить значение, которое не поддерживается видеокартой — то соответственно сглаживание работать не будет и его включение ничего не даст.

Включить / выключить вертикальную синхронизацию можно методом:

```
Procedure VertSynh(Enable:boolean);
```

Стиль заливки графических примитивов (треугольников, четырёхугольников и т.д.) устанавливается при помощи метода:

```
Procedure SetFill(Mode:TGLFill);
```

где, Mode может принимать значения glPoint, glLine, glFill соответственно - точки, линии, сплошная заливка.

Следует так же обратить внимание на то, что все цвета задаются в режиме RGBA, и следовательно имеют альфаканал. В GLEngine имеется возможность указать как работать с альфаканалом, для этого следует воспользоваться методом:

```
Procedure SwichBlendMode(BlendMode:TGLBlendMode);
```

где TGLBlendMode=(bmAdd,bmNormal).

bmNormal — режим нормального смешивания и является режимом «по умолчанию».

bmAdd – режим добавления (умножения цветов).

## 3. Графические примитивы

### 3.1 Точки

Для вывода на экран точки используется метод:

```
Procedure Point(x1,y1:single);
```

где x1 и y1 – x и y координаты точки соответственно.

Так же для точки можно установить её размер:

```
Procedure PointSize(Size:single);
```

где Size – это размер точки в «виртуальных пикселях».

При выводе точки размером например 4, в разных задачах это может быть как круг диаметром 4, так и квадрат со стороной 4. Что бы определить однозначно, что выводить используется метод:

```
Procedure PointSmooth(Enable:Boolean);
```

где Enable – указывает режим сглаживания точки при масштабировании. Если Enable = True, то на экране будет круг, если Enable = False – соответственно квадрат.

### 3.2 Линии

Что бы нарисовать линию необходимо вызвать метод:

```
Procedure Line(x1,y1,x2,y2:single);
```

где x1,y1,x2,y2 — это координаты начальной (x1,y1) и конечной (x2,y2) точек. Так же можно управлять шириной выводимых линий:

```
Procedure LineWidth(W:single);
```

где W – ширина линии в пикселях.

Так же можно осуществить вывод градиентной линии, у которой цвет будет плавно меняться от цвета начальной точки до цвета конечной.

```
Procedure LineGrad(x1,y1,x2,y2:single;Color1,Color2:TGLColor);
```

где x1,y1,x2,y2 — это координаты начальной (x1,y1) и конечной (x2,y2) точек, а Color1 и Color2 – цвет начальной и конечной точки соответственно.

Для того, чтобы сделать линию штрихпунктирной или задать любой другой стиль отображения линий имеется метод:

```
Procedure SetLineStripple(Factor: GLint; Line: string);
```

Для отмены стиля и возвращения к нормальному отображению линий вызывается метод:

```
Procedure UnSetLineStripple;
```

### Пример

```
GLE.SetLineStripple(1, '----    ----    ');  
GLE.Line(40,20,80,20);  
gle.UnSetLineStripple;
```

### 3.3 Треугольники

Треугольники задаются тремя точками своих вершин (X1,Y1) (X2,Y2) (X3,Y3):

```
Procedure Triangle(x1,y1,x2,y2,x3,y3:single);
```

Можно вывести треугольник закрашенный градиентом, для этого необходимо ещё указать цвет каждой вершины:

```
Procedure TriangleGrad(x1,y1,x2,y2,x3,y3:single;c1,c2,c3:TGLColor);
```

где C1,C2,C3 – соответственно цвета первой, второй и третьей вершины.

### 3.4 Четырёхугольники и прямоугольники

Для задания прямоугольника достаточно указать две точки, по которым он будет построен

```
Procedure Bar(x1, y1, x2, y2:single);
```

или можно указать координаты середины(X,Y) прямоугольника, его длину (W) и ширину (H), а так же угол поворота(angle):

```
Procedure Bar(x, y, w, h, angle:single);
```

Для не закрашенного прямоугольника (только рамочка, вне зависимости от настроек SetFill):

```
Procedure Rectangle(x1, y1, x2, y2:single);
```

Четырёхугольник задаётся 4 вершинами:

```
Procedure Bar(x1, y1, x2, y2, x3, y3, x4, y4:single);
```

Можно задать каждой вершине цвет, и тогда получим четырёхугольник с градиентной заливкой:

```
Procedure BarGrad(x1,y1,x2,y2,x3,y3,x4,y4:single;c1,c2,c3,c4:TGLColor);
```

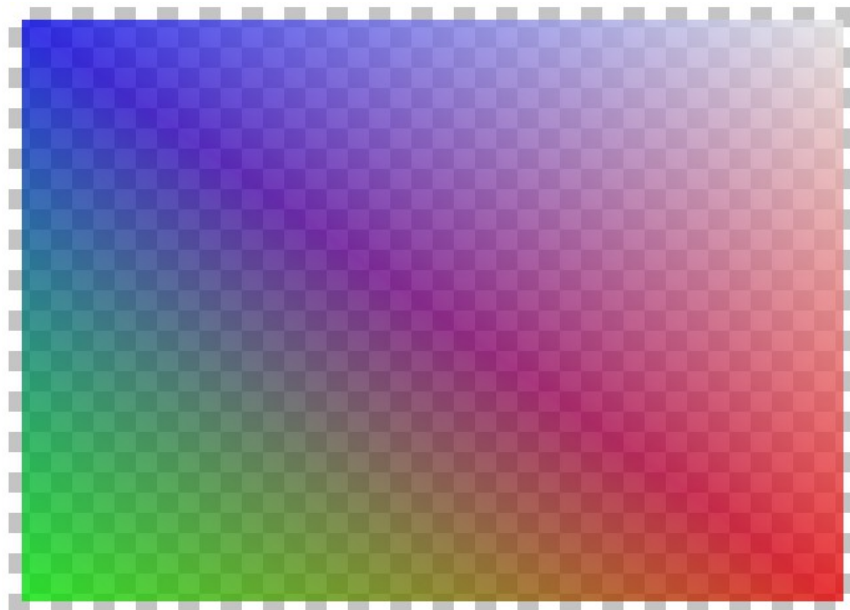


Рисунок 3.4.1 «Пример выполнения

Gle.BarGrad(10,10,10,470,630,470,630,10,gle.ColorGL(1,0,0,0.9),gle.ColorGL(0,1,0,0.9),gle.ColorGL(0,0,1,0.9),gle.ColorGL(1,1,1,0.9));»



### 3.5 Круги, окружности и эллипсы

Реализуются одним методом:

```
Procedure Ellipse(x,y,r1,r2,AngleRotate:single;n:integer);
```

где: x,y – координаты центра  
r1,r2 – радиусы эллипса  
AngleRotate — угол поворота эллипса в градусах  
n – количество сегментов из которых состоит эллипс

### 3.6 Текст

Вывести текст в нужной позиции позволяет метод:

```
Procedure TextOut(x,y:single; text:string);
```

где: x,y – координаты начала строки  
text – сама строка выводимого текста

Для изменения шрифта и размера текста используется метод

```
Procedure SetTextStyle(NameFont:string; size:integer);
```

где: NameFont – название шрифта, например «Times New Roman»  
size – размер шрифта.

### 3.7 Прочие примитивы

Так же доступны примитивы, которые несколько упрощают жизнь разработчику:

```
Procedure Bolt(x1,y1,x2,y2:single);
```

Этот метод выводит извилистую линию от точки (x1,y1) до точки (x2,y2).

```
Procedure Arrow(x1,y1,x2,y2, size,angle:single;Solid:Boolean);
```

Стрелочка от точки (x1,y1) в точку (x2,y2). Size – размер самой стрелочки, angle – угол наклона линий, образующих стрелочку, Solid – если равен True – стрелочка закрашенная, иначе — нет.

## 4. Работа с изображениями

### 4.1 Загрузка и создание

Прежде чем выводить изображение его необходимо загрузить из файла или создать. Работа с изображениями происходит по указателям. Указателем выступает переменная типа Cardinal.

#### Пример

```
Var  
im:Cardinal;
```

GLEngine может работать с файлами следующих типов: TGA BMP JPEG PNG GIF. Загрузка изображения осуществляется вызовом метода:

```
function LoadImage(Filename: String; var Texture : Cardinal;  
LoadFromRes : Boolean) : Boolean;
```

где: Filename – строка содержащая путь к загружаемому файлу  
Texture – переменная которая будет содержать указатель на загруженное изображение  
LoadFromRes – если равен True загрузка происходит из ресурсов, иначе из внешнего файла.  
Помимо загрузки из файла возможна загрузка из TBitMap:

```
procedure AddBMPImage (Bmp:TBitMap;var Texture : TGLuint);
```

однако следует иметь в виду, что BMP должен быть с 32-битной глубиной цвета.  
Так же можно создать «пустое» изображение:

```
Function CreateImage (w,h:integer):Cardinal;
```

где: w и h – это ширина и высота создаваемого изображения соответственно. Следует иметь ввиду, что имеются аппаратные ограничения на размер изображений. Как правило это 4096x4096 хотя могут быть и другие.

Следует отметить что после завершения работы с изображением, необходимо освободить занимаемую им память методом:

```
Procedure FreeImage (var Texture : Cardinal);
```

Освобождением памяти занимается драйвер вашей видеокарты.

### 4.2 Отрисовка изображений

Для вывода изображения используется метод:

```
Procedure DrawImage (x,y,w,h,Angle:single;Center:boolean;Image:Cardinal);
```

где: x,y – координаты где будет располагаться изображение  
w,h – ширина и высота выводимого изображения  
Angle – угол поворота изображения  
Center – если принимает значение True, то изображение будет центроваться (центр изображения будет находится в координатах X,Y), иначе — X,Y – координаты нижнего левого угла.  
Image — переменная изображения.

Следует отметить что если перед тем как вывести изображение вызвать SetColor, то изображение будет выводиться согласно заданному цвету.

При выводе большого количества одинаковых изображений, лучше использовать связку методов SetCurrentImage и DrawCurrentImage

```
Procedure SetCurrentImage (Image:Cardinal);
```

где: Image – изображение, которое мы будем выводить.

```
Procedure DrawCurrentImage (x,y,w,h,Angle:single;Center,tile:boolean);
```

метод аналогичный DrawImage, только для вывода использует изображение Image, установленное методом SetCurrentImage.

### 4.3 Рисование на изображении (FBO)

По-умолчанию рисование происходит на компоненте, чей хендл был указан при инициализации GLEngine2D. Что бы рисовать не на компоненте, а на изображении используют метод:

```
Procedure BeginRenderToTex (Image:Cardinal; w,h:glint);
```

где: Image – изображение на котором будет происходить рисование.  
W,H – ширина и высота этого изображения соответственно.  
Что бы вернуться к рисованию на компоненте надо вызвать метод:

```
Procedure EndRenderToTex;
```

#### Пример

```
var  
    CreateTex:Cardinal;  
...  
CreateTex:=GLE.CreateImage (640,480);  
GLE.Resize (640,480);  
GLE.BeginRenderToTex (CreateTex,640,480);  
GLE.BarGrad (10,10,10,470,630,470,630,10,gle.ColorGL (1,0,0,0.9),gle.ColorGL (0,1,0,0.9),gle.ColorGL (0,0,1,0.9),gle.ColorGL (1,1,1,0.9));  
gle.EndRenderToTex;
```

### 4.4 Сохранение

Для сохранения изображения можно воспользоваться одним из методов:

```
Procedure SaveImage (FileName:string;var Texture : Cardinal);
```

где: FileName – строка, содержащая адрес файла, в который будет сохранено изображение Texture. Изображение будет сохранено в формате BMP с глубиной цвета 32.

```
Procedure SaveImageAsPNG (Filename:String; Im:Cardinal);
```

где: FileName – строка, содержащая адрес файла, в который будет сохранено изображение Im. Изображение будет сохранено в формате PNG с глубиной цвета 32.

```
Procedure GetBMP32FromImage (Im:Cardinal;var BMP32:TBitMap);
```

где: Im – изображение которое будет сохраняться в BMP32. BMP32 должен быть создан и уже существовать. Формат цвета и размеры изменяются автоматически.

#### Пример

```
var  
    bmp:TBitMap;  
...  
bmp:=TBitMap.Create;  
gle.GetBMP32FromImage (CreateTex,bmp);  
image1.Picture.Assign (bmp);  
bmp.Free;
```

## 5. Работа с шейдерами

### 5.1 Введение (взято с [gamedev.ru](http://gamedev.ru) и [elite-games.ru](http://elite-games.ru))

GLSL (OpenGL Shading Language) - glSLang - новый язык высокого уровня для создания фрагментных и вершинных шейдеров. В отличие от HLSL/Cg он создавался в расчете на будущее железо, поэтому теоретически он намного мощнее. В частности, GLSL много взял от RenderMan Shading Language. Существует два вида шейдеров:

**Вершинный** вызывается для каждой вершины. Ему на вход поступает координата вершины в мировой системе координат, текстурные координаты, нормали и еще много разных параметров (в том числе и любые другие, которые вы захотите). Скорость выполнения вершинного шейдера на процессоре видеокарты (GPU) довольно высока, по крайней мере, превышает скорость CPU в разы. Но пиксельные шейдеры – вот настоящая скорость...

**Пиксельный** вызывается для каждого (внимание!) пикселя. Скорость его настолько высока, что у меня, например, на GeForce 6600 пиксельный шейдер, на котором сделано гауссово размытие, замедляет счетчик FPS всего лишь на 20 кадров в секунду. Ни один CPU не сравнится по скорости с GPU. Пиксельному шейдеру поступают на вход измененные данные из вершинного шейдера и из вашей программы, а на выход должен выйти лишь один параметр – цвет пикселя.

### 5.2 Загрузка и создание

Прежде чем работать с шейдерами их необходимо создать или загрузить из файла. Работа с шейдерами происходит, как и с изображениями, по указателям. Указателем выступает переменная типа Cardinal.

#### Пример

```
Var  
s,b:cardinal;
```

Для загрузки шейдеров из файлов используется метод:

```
Function ShaderCreate (FragmentShaderFileName,VertexShaderFileName:String) :  
Integer;
```

где: FragmentShaderFileName и VertexShaderFileName — строки, содержащие адреса файлов с текстом шейдеров.

#### Пример

```
s:=Gle.ShaderCreate ('b.fp', 'b.vp');  
b:=Gle.ShaderCreate ('blur.fp', 'blur.vp');
```

где: FragmentShaderFileName и VertexShaderFileName — строки, содержащие адреса файлов с текстом шейдеров.

### 5.3 Использование

Для того что бы использовать загруженный шейдер используется метод:

```
Procedure ShaderStart (Shader:Integer) ;
```

Что бы вернуться в нормальный режим рисования и отменить действия шейдера используется метод:

```
Procedure ShaderStop (Shader:Integer) ;
```

#### Пример

```
gle.ShaderStart(s);  
    gle.Bar(170,165,310,310,0);  
gle.ShaderStop(s);
```

Продолжение следует...