The Miracle of Creation

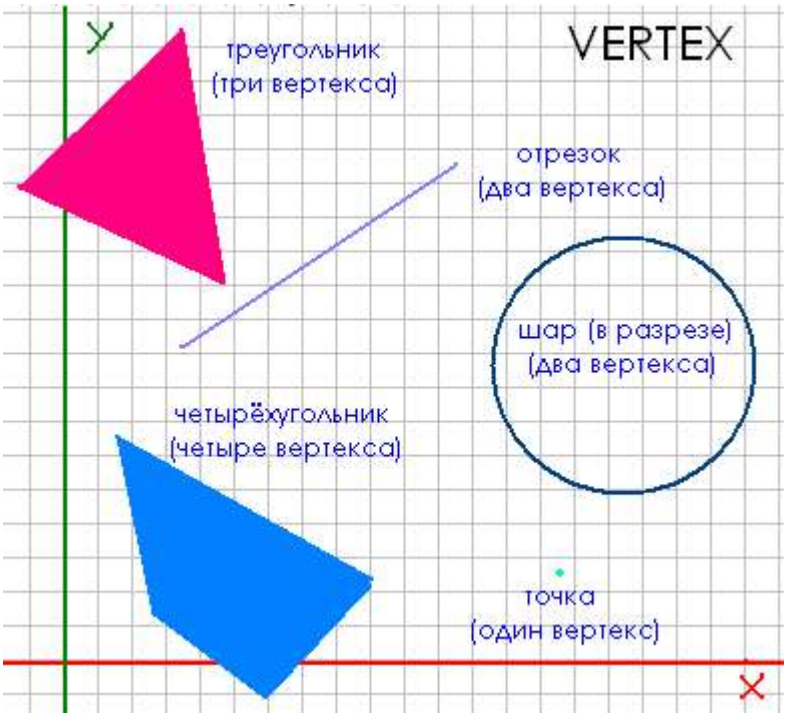
> В начале сотворил Бог небо и землю. Земля же была безвидна и пуста, и тьма над бездною, и Дух Божий носился над водою."
>
> Первая книга Моисея

В предыдущей главе (the.first.blood) мы прорубили окно в OpenGL. Совершенно пустое и полное загадок. При компиляции примера у Вас должны были возникнуть ошибки, которые, будем надеяться, Вы с успехом исправили. Для тех, кто пренебрёг советом из предыдущей главы, прозвучал тревожный звоночек – господа, примеры нужно изучать внимательнее.

Vertex отрыва

Точки бывают разные. Wasm точка ru, точка зрения, долговременная огневая точка (ДОТ), точка на листе бумаги. Можно легко запутаться, какое значение слова «точка» имееют в виду в каждом отдельном случае. Поэтому, в компьютерной графике существует такое понятие как vertex – это математический объект, который определяется тремя основными свойствами: координатой x, координатой y и координатой z в трехмерном пространстве виртуальной реальности.

Вертексы являются математическим скелетом, на который OpenGL, следуя нашим инструкциям, наращивает пиксельное мясцо. Так, если мы хотим построить отрезок, то передаем OpenGL только информацию о свойстве отрезка (вертексы концов), а не множество всех его точек. Это множество точек нашего отрезка OpenGL вычисляет и рисует самостоятельно, на математический скелет из двух вертексов наращивает плоть, и на экране монитора появляется отрезок. Минимальное кол-во вертексов необходимое для описания треугольника равно трем. Для описания отрезка необходимо два вертекса, а для точки – ровно один вертех.



Логично предположить, что при передаче в OpenGL вертексов для постороения сцены следует также сообщать о фигурах, которые эти вертексы описывают. Ведь три вертекса в разных ситуациях могут описывать: три точки, отрезок и точку, треугольник или же ломаную линию.

```
; MASM
; Передача информации для построения треугольника в OpenGL

; Мы сообщаем OpenGL, что собираемся передавать вертексы,
; которые описывают геометрическую фигуру «треугольник»
invoke glBegin, GL_TRIANGLES

; Первый вертекс - описывает первую вершину нашего треугольника   (3.45f, 24.4f, 0.23f),
; то есть (x, y, z) в виртуальном математическом пространстве.
invoke glVertex3f, 3.45f, 24.4f, 0.23f

;  Второй вертекс - описывает вторую вершину нашего треугольника   (-23.4f, 0.01f, 10.23f)
invoke glVertex3f, -23.4f, 0.01f, 10.23f

;  Третий вертекс - описывает третью вершину нашего треугольника    (-0.4f, -0.002f, -10.23f)
invoke glVertex3f, -0.4f, -0.002f, -10.23f

; Сообщаем, что передача вертексов закончена. В математическом виртуальном 3D пространстве OpenGL появляется объект «треугольник»
invoke glEnd
```

Поменяем константу GL_TRIANGLES на GL_POINTS, и эти три наших вертекса будут описывать три точки, а не один треугольник. Если мы захотим в нашем

примере изобразить три треугольника, нам будет достаточно добавить между связкой glBegin/glEnd ещё 6 вертексов (по 3 на треугольник).

Имя функции glVertex3f говорит само за себя: передаются три координаты вертекса, тип данных GLfloat. OpenGL представляет нам богатые возможности в способе передачи вертексов. Если наша фигура лежит в плоскости XY (т.е. координаты z её вертексов равна нулю), мы можем использовать функцию glVertex2f.

Вертексы понятны OpenGL только в рамках парадигмы glBegin/glEnd, иначе они не будут иметь никакого смысла. OpenGL не обладает телепатическими способностями и, если ему не сообщить название геометрической фигуры (GL_TRIANGLES), которые мы хотим описать, он не в жизнь не догадается.

Петька, приборы!!?
Василий Иванович, двадцать!
Что «двадцать»?
А что «приборы»??!!

There is no spoon.

Что же представляет из себя трехмерная виртуальная реальность? Ответить на этот вопрос чрезвычайно трудно. Давайте пока оставим попытки определения метафизического смысла виртуальной реальности философам, и заострим всё наше внимание на очевидном факте – виртуальная реальность это прежде всего иллюзия, построенная на знаниях человека об окружающем мире, его физических законах. Эксплуатируя знание о восприятии человеком реального мира, удается обмануть наш мозг настолько, что начинаешь сочувствовать кучке разноцветных треугольников (например, герою вашей любимой 3D компьютерной игры) и с чистым сердцем давить на кнопку, которая никогда не существовала в природе.

Как же перенести наши фантазии в виртуальный мир, который по сути является некой математической моделью? Очевидно, что необходимо перевести фантазию на язык математики, которую отлично понимает OpenGL и, более того, охотно воспроизводит на экране нашего монитора. Это необходимо, но …

недостаточно. Язык математики безусловно богат и универсален, но даже его не хватает, чтобы дословно перевести на него язык фантазии без потери смысла. «Trifles go to make perfection, and perfection is no trifle» (c) Michaelangelo. Тонко подмеченные и переданные в виртуальность нюансы реальности заставляют нас сопереживать разноцветным треугольникам на экране. Безукоризненно воспроизведенная математическая модель реальности это безусловно настоящий полёт, а переданное настроение – его высший пилотаж. Об этом нужно помнить.

- Видишь суслика?
- Нет.
- И я не вижу, а он есть.
(c) «ДМБ»

Через тернии к звёздам

Давайте добавим эти строчки в нашу первую программу из предыдущей главы:

```
; (Строчка из first.blood.asm, после которой мы попытаемся нарисовать треугольник)
.ELSEIF uMsg ==  WM_PAINT

; Мы сообщаем OpenGL, что собираемся передавать вертексы,
; которые описывают геометрическую фигуру «треугольник»
invoke glBegin, GL_TRIANGLES

; Первый вертекс - описывает первую вершину нашего треугольника   (3.45f, 24.4f, 0.23f),
; то есть (x, y, z)
invoke glVertex3f, 3.45f, 24.4f, 0.23f

;  Второй вертекс - описывает вторую вершину нашего треугольника    (-23.4f, 0.01f, 10.23f)
invoke glVertex3f, -23.4f, 0.01f, 10.23f

;  Третий вертекс - описывает третью вершину нашего треугольника     (-0.4f, -0.002f, -10.23f)
invoke glVertex3f, -0.4f, -0.002f, -10.23f

; Сообщаем OpenGL, что передача вертексов закончена
invoke glEnd
```

Компилируем и получаем кучу ошибок “real or BCD numbers are not allowed!” Вот такой вот облом-с. Только, понимаешь, захотели сотворить умное-доброе-вечное, а злой invoke не позволяет нам использовать вещественные числа в качестве аргумента. Что ж, придётся кормить OpenGL с вилки (ложки, как

известно, у нас нет). «Не доверяй технике, более сложной, чем нож или вилка» (с) Роберт Энсон Хайнлайн

```
; MASM
; Хитрый макрос.
fpush   MACRO arg
        push 12345678h
        ORG $-4
        REAL4 arg
        ENDM
```

Исправляем нашу программу

```
; MASM
invoke glBegin, GL_TRIANGLES

; - "Первый", готов?
fpush 0.23f
fpush 24.4f
fpush 3.45f
; - Готов!
; - Паааашел!
call glVertex3f

; - "Второй", готов?
fpush 10.23f
fpush 0.01f
fpush -23.4f
; - Готов!
; - Паааашел!
call glVertex3f

; - "Третий", готов?
fpush -10.23f
fpush -0.002f
fpush -0.4f
; - Готов!
; - Паааашел!
call glVertex3f

; - "Четвёртый", готов?
; - Не готов!
; - Паааашел! … первый "готов"…
invoke glEnd
```

Всё скомпилилось превосходно, а в окне по-прежнему пустота. В чем же тут дело? Давайте порассуждаем. Мы хотели увидеть на экране треугольник, это наша примитивная фантазия. Для этого мы перевели её на язык математики, описали фигуру «треугольник» при помощи трех вертексов и загрузили в OpenGL. Такое математическое описание нашей фантазии понятно OpenGL, в его виртуальном математическом 3D пространстве появился единственный пока объект – наш треугольник.

Экран монитора это плоское изображение, а виртуальное пространство трехмерно. Заглянем в наш альбом с фотографиями. Снимки плоские (проверьте сами, если не верите), а изображения являются точным изображением кусочка нашей 3D реальности. Хотим сфотографировать дом – ловим его в видоискателе фотоаппарата и нажимаем на спусковую кнопку. На фотографии появляется плоское изображение дома, свет создает проекцию реального трехмерного объекта на плёнку. При «фотографировании» трехмерной виртуальной реальности OpenGL делает проекцию трехмерных математических объектов на плоскость экрана нашего монитора.

Как же получить на экране монитора фотографию нашего треугольника? Для начала нужно поймать его в «видоискатель». Для этого воспользуемся функцией OpenGL glOrtho.

```
; C
void glOrtho(  GLdouble left,
               GLdouble right,
               GLdouble bottom,
               GLdouble top,
               GLdouble near,
               GLdouble far )
```

С её помощью можно выбрать кусок трехмерной виртуальной реальности, с которого мы хотим получить проекцию на экран монитора.

```
; MASM
; (Строчка из first_blood.asm, после которой мы добавляем комманду glOrtho)
        invoke CreateOGLWindow

; Захватываем в «видоискатель» кусок трехмерной виртуальной реальности,
; да побольше – кубик пространства размером
; 200.0f x 200.0f x 200.0f (ширина x высота x глубина). Проверьте, попадают ли все
; вертексы нашего треугольника в этот «сачок».
        invoke glOrtho, -100.0f, 100.f, -100.0f, 100.0f, -100.f, 100.0f
```

Компилируем и … получаем кучу уже знакомых ошибок “real or BCD numbers are not allowed!” Бодро исправляем invoke на fpush/call, успешно компилируем и запускаем приложение, которое тут же рушится от критической ошибки.

Дело в том, что функция glOrtho ожидает шесть чисел типа GLdouble (QWORD). Мы же отправляем в стек только половину (6 x DWORD) и заслуженно получаем ошибку. Можно использовать, например, такой макрос.

```
; MASM
; Безхитросный макрос для загрузки QWORD в стек

dpush                   MACRO       arg
                        LOCAL lbl, double
                        jmp lbl
                        double REAL8 arg
                        lbl:
                        PUSH DWORD PTR [double+4]
                        PUSH DWORD PTR [double]
                        ENDM

[skip]

; Вызов функции glOrtho приобретает такой компактный вид
dpush  100.0f
dpush -100.0f
dpush  100.0f
dpush -100.0f
dpush  100.0f
dpush -100.0f
call glOrtho
```

Или использовать invoke.

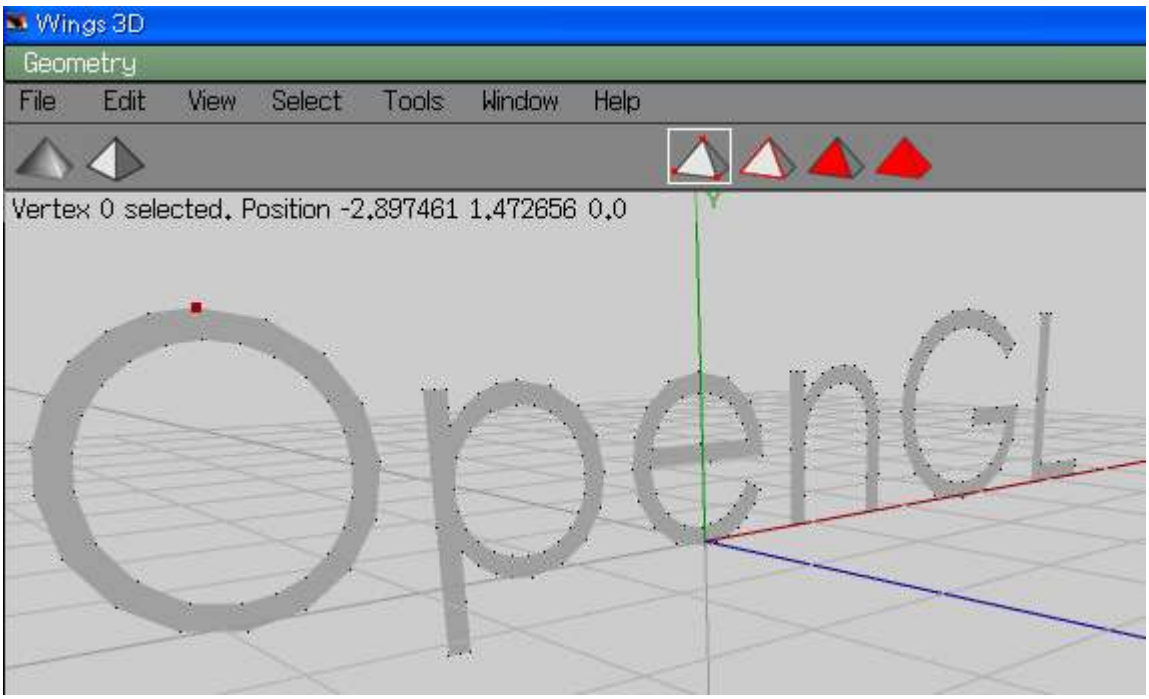
```
; MASM
; Добавим две переменные в раздел .DATA

dbl_x REAL8 100.0f
dbl_xneg REAL8 -100.0f

[skip]

; и исправим код вызова glOrtho
invoke glOrtho,  DWORD PTR dbl_xneg, DWORD PTR dbl_xneg+4,\
                 DWORD PTR dbl_x, DWORD PTR dbl_x+4, \
                  DWORD PTR dbl_yneg, DWORD PTR dbl_yneg+4, \
                 DWORD PTR dbl_y, DWORD PTR dbl_y+4, \
                DWORD PTR dbl_zneg, DWORD PTR dbl_zneg+4, \
                 DWORD PTR dbl_z, DWORD PTR dbl_z+4
```

Теперь остаётся только нажать на спусковую кнопку (glFlush) и любоваться фотографией треугольника.

Очень одинокий белый треугольник в пустом виртуальном пространстве.

Spread Your Wings 3D!

Вы готовы изобразить что-нибудь посерьёзнее? Для этого нам необходима математическая модель изображения для импортирования её в виртуальную реальность OpenGL. Не будем откладывать это в долгий ящик и скачаем программу Wings 3D (link to http://www.wings3d.com/). За 2МБ сетевого трафика мы получаем простой в освоении и крайне необходимый инструмент для конструирования одно-, двух- и трехмерных объектов. К тому же как и OpenGL это инструмент мультиплатформенный (Windows. Mac OS X, GNU/Linux и другие *nix), нетребовательный к ресурсам и при всём при этом абсолютно бесплатный.



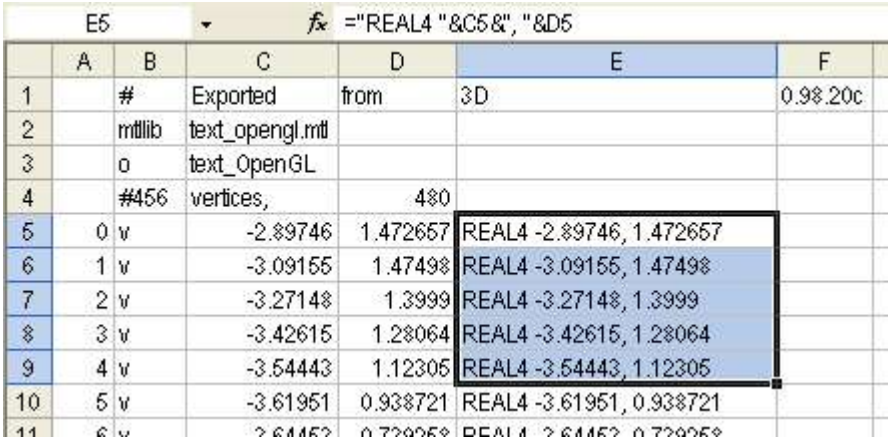
Wings3D позволяет нам интерактивно конструировать объекты в 3D пространстве, и на лету оценивать результат своей работы. Например, математическую модель плоской надписи «OpenGL» в шрифте Century Gothic можно автоматически построить за пару секунд. Сколько времени нам понадобится, чтобы расставить вертексы (показаны чёрными точками) вручную? А на листе миллиметровки? А в уме?

Для построения надписи «OpenGL» в нашей программе, необходимы координаты её вертексов. На скриншоте выделен красным цветом вертекс под номером 0, с координатами x, y, z . Можно, конечно, «пробежать» по всем вертексам надписи «OpenGL» и выписать их координаты на бумажку, чтобы затем использовать в нашем приложении. Но «мы пойдем другим путём» (с) В. И. Ленин. Wings3D поддерживает очень простой текстовый формат представления 3D моделей в трехмерной виртуальной реальности – Wavefront OBJ.

```
# Exported from Wings 3D 0.98.20c
mtllib text_opengl.mtl
o text_OpenGL
#456 vertices, 480 faces
v -2.89746 1.472657 0.0     ------→ координаты вертекса с номером 0.
v -3.09155 1.47498 0.0
v -3.27148 1.39990 0.0
```

Пока нас интересуют только вертексы, они отмечены в файле буквой “v”. Первая буква «O» описывается вертексами с индексами от 0 до 17 (внешняя

«окружность»), и от 18 до 39 (внутренняя «окружность»). В этом можно легко убедиться, если провести курсором по вертексам буквы «O». Подобным образом можно найти все вертексы всех букв модели «OpenGL» и скопировать их в нашу программу из файла Wavefront OBJ. Можно ускориться, используя Excel. Открыть Wavefront OBJ, расставив разделители ячеек на месте пробелов. Удалить колонку с координатами z (они равны нулю, так как надпись «OpenGL» лежит в плоскости XY), добавить колонку с индексами вертексов и написать простейшую формулу, чтобы комфортно скопировать данные в наш ASM файл.



the.miracle.of.creation.asm

Возьмём за основу исходник first_blood.asm из предыдущей главы. Добавим три прототипа функции RenderGLScene, InitOGL и загадочную LoadVertexData.

```
; MASM
; С помощью этой функции мы будем отрисовать нашу сцену в OpenGL
RenderOGLScene PROTO

; Эта функция необходима нам для настройки OpenGL  перед его использованием
InitOGL PROTO

; При помощи этой функции мы будем загружать нашу модель в виртуальную
; реальность OpenGL
LoadVertexData PROTO :DWORD
```

Теперь давайте добавим в наш исходник 2D модель текста «OpenGL», которую мы будем загружать при помощи функции LoadVertexData. Для этого в секцию .DATA поместим результаты нашего моделирования в Wings3D. Например, можно структурировать информацию о модели следующим образом:

```
; MASM
; VertexData: вертексное описание модели плоского текста «OpenGL»

; Число вертексов, описывающее замкнутую кривую
; внешней «окружности» буквы «O»
VertexData DWORD 18
; 18 вертексов. Заданны координатами x и y (z равно нулю)
REAL4 -2.89746, 1.472657
REAL4 -3.09155, 1.47498
[skip]
REAL4 -2.5979, 1.44556

; Число вертексов, описывающее замкнутую кривую
; внутренней «окружности» буквы «O»
DWORD 22
; 22 вертекса. Заданны координатами x и y (z равно нулю)
REAL4 -2.89063, 1.35938
[skip] буквы 'p', 'e', 'n', 'G', 'L'

; Кривых в модели больше нет.
DWORD 0
```

Начнем разбираться с LoadVertexData. Как видно на скриншоте Wings3D, все буквы могут быть легко отрисованы замкнутыми кривыми. Первая буква «O» состоит из двух замкнутых кривых – внешняя «окружность» и внутренняя. Чтобы OpenGL понял, что мы загружаем вертексную информацию для построения замнутой кривой, нам необходимо сказать об этом при вызове функции glBegin.

```
; MASM

LoadVertexData proc GeometricShape:DWORD

; Устанавливаем счетчик вертексов (ecx) и адрес указывающий на первую пару
; координат вертекса (eax) некой геометрической фигуры  (определяется
; переменной GeometricShape)
lea eax, VertexData
mov ecx, [eax]
add eax, 4

; начало цикла загрузки всех геометрических фигур, из которых состоит наша модель
; «OpenGL» в шрифте Century Gothic
load_next_shape:

; Всё внимание на push/pop. Регистры после вызовов функций OpenGL скорее всего
; изменят свои значения.
push eax
push ecx
; Мы сообщаем OpenGL, что собираемся загружать вертексы, описывающие
; некую геометрическую фигуру: GL_POINTS или GL_LOOP_LINE или GL_TRIANGLES
invoke glBegin, GeometricShape

        ; восстанавливаем счетчик и указатель
        pop ecx
        pop eax
```

```
        ; начало цикла, загружающего текущую геометрическую фигуру
        ; (одну замкнутую линию) из нашей (нашей!!!!) 2D модели
        @@:

        ; Сохраняем нужные нам регистры, загружаем в OpenGL очередной вертекс
        ; и восстанавливаем счетчик и указатель
        push eax
        push ecx
        invoke glVertex2f, DWORD PTR [eax], DWORD PTR [eax+4]
        pop ecx
        pop eax
        ; готовимся к передаче следующего вертекса из текущей геометрической фигуры
        dec ecx
        add eax, 8
        .IF ecx != 0
                jmp @B
        .ENDIF
; Вертексы, описывающие текущую фигуру, закончились. Сообщаем об этом
; OpenGL, предварительно сохранив указатель
push eax
invoke glEnd

; Переходим к следующей геометрической фигуре.
pop eax
mov ecx, [eax]
.IF ecx !=0
        add eax, 4
        jmp load_next_shape
.ENDIF

; Все геометрические фигуры, из которой состоит наша модель, загружены в
; OpenGL. Выходим.
ret
LoadVertexData endp
```

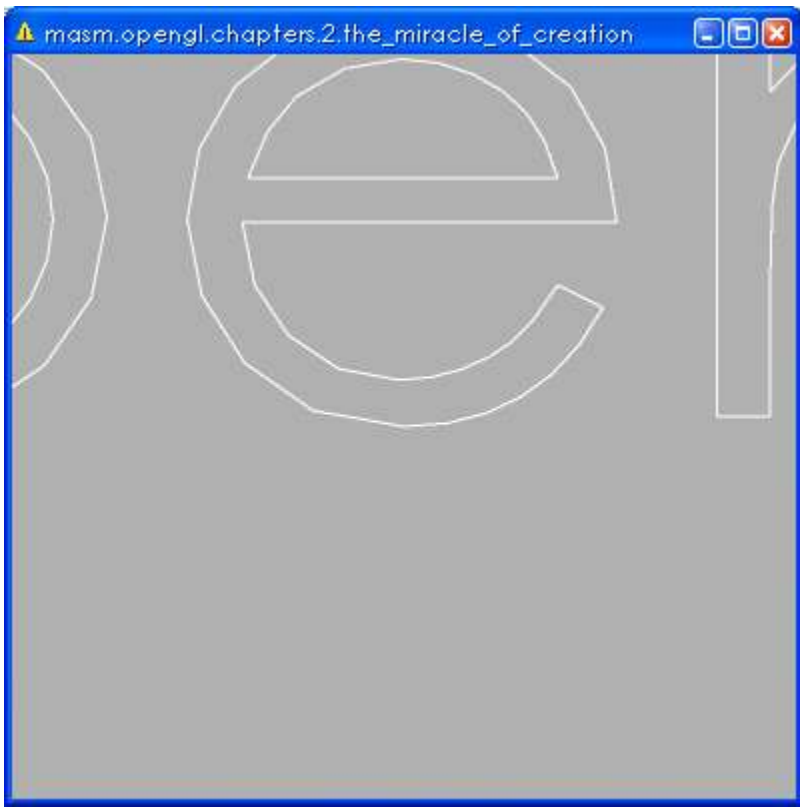
Вот ради чего всё, собственно, затевалось:

```
; MASM
RenderOGLScene proc
; Берём чистый листок бумаги (очищаем экран)
invoke glClear, GL_COLOR_BUFFER_BIT

; Загружаем нашу модель в виртуальное пространство OpenGL
invoke LoadVertexData, GL_LINE_LOOP

; Делаем снимок
invoke glFlush
ret
RenderOGLScene endp
```

Исправляем код в обработчике сообщения windows WM_PAINT

```
; MASM
.ELSEIF uMsg ==  WM_PAINT
        invoke RenderOGLScene
.ENDIF
```

Компилируем и запускаем the.miracle.of.creation.exe



Очевидно, что пред съемкой необходимо настроить объектив камеры OpenGL, чтобы на «фотографию» влезли все буквы. Вот для этого нам нужна подпрограмма InitOGL.

```
InitOGL proc

; Выбираем цвет фона нашей фотографии
fpush 0.5
fpush 0.6
fpush 0.2
fpush 0.6
call glClearColor
; Очищаем экран выбранным цветом
invoke glClear, GL_COLOR_BUFFER_BIT

; настраиваем видоискатель
invoke glOrtho,DWORD PTR dbl_xneg, DWORD PTR dbl_xneg+4, \
                DWORD PTR dbl_x, DWORD PTR dbl_x+4, \
               DWORD PTR dbl_yneg, DWORD PTR dbl_yneg+4, \
                DWORD PTR dbl_y, DWORD PTR dbl_y+4, \
               DWORD PTR dbl_zneg, DWORD PTR dbl_zneg+4, \
                DWORD PTR dbl_z, DWORD PTR dbl_z+4
ret
InitOGL endp
```

… где …

```
; MASM
.DATA
dbl_x REAL8 5.5f
dbl_xneg REAL8 -5.5f
dbl_y REAL8 5.0f
dbl_yneg REAL8 -5.0f
dbl_z REAL8 1.0f
dbl_zneg REAL8 -1.0f
```

Настраиваем OpenGL после успешного создания окна:

```
; MASM

invoke CreateOGLWindow
.IF eax == FALSE
	invoke CloseOGLWindow
	ret
.ENDIF

invoke InitOGL
```

Компилируем, запускаем, наслаждаемся.

Информация к размышлению:

1) Какой размер в памяти компьютера занимает GLfloat?
2) В чем отличие функций PeekMessage и GetMessage?
3) Как можно вывести текст в OpenGL?

Задание на дом:

1] Прочитайте описание всех функций OpenGL из этой главы в мануале "OpenGL Programming Guide" aka "The Red Book"
(link to http://fly.cc.fer.hr/~unreal/theredbook/)

2] Постройте изображении нашей модели в виде точек, треугольников

3] Научитесь работать с программой Wing3D

4] Постройте плоскую модель символа "дзен" в Wing3D (предмет изучения следующей главы)