Window – Place

Интерактивная среда разработки многооконных (многозадачных) приложений в контекстно-зависимой трехмерной графике OpenGL с поддержкой стековых наложений графических фрагментов и возможностью задействования виртуальных управляющих процедур C++.

Window:Place – пакет базовых определений операций[1] в контекстной среде программирования C++ для трехмерной научной графики на основе OpenGL. Объектно-ориентированный комплекс создает интерфейс между программой, операционной системой и внешними устройствами: компьютерными часами и интервальным таймером; внутренними растровыми и системными векторными шрифтами; графическим терминалом; клавиатурой и указателем «мышь»; другими внешними устройствами (*измерительной телеметрией*). Производный класс **Window**:Place управляет одним из активных окон с собственным контекстом OpenGL и доступом к таймеру и клавиатуре. Базовый класс **Place** и варианты дополнительных объектов на его основе формируют стековые наложения графических площадок/фрагментов на поверхности окна Window, для которых раздельно устанавливаются режимы отображения с контролем исполнения трехмерной графики с автоматическими перерисовками для плоских картинок, текстовых отчётов, меню и справок. Курсор (мышь) передаёт координаты своего местоположения на верхнюю/видимую площадку Place. Полиморфизм производных классов для прикладных вычислительных объектов, допускает подмену базовых виртуальных функций, что может быть полезным для сквозной перенастройки графических изображений или ускорения вычислений.

Ключ #define GLFW переключает прямое обращение к процессам Windows-WGL на независимую от операционной системы графическую среду GLFW-OpenGL.

[1] «Контекстная графика» *(Контекстно-зависимая среда построения трехмерной графики OpenGL с использованием виртуальных процедур C++ и многооконного интерфейса Windows со стековым наложением графических и текстовых фрагментов). ©Храмушин В. Н., Сахалинский государственный университет.* Роспатент: *Свидетельство о государственной регистрации* № 2010615850, 2010.09.08. Заявка 2010614191, 2010.07.13.

Список основных процедур с указанием параметров Window::Place – OpenGL

Window:: *//! блок управления графическим окном*

```
Window( Title, X,Y, Width,Height )      // заголовок и местоположение окна
        Xpm( X ), Ypm( Y ) // макросы размерения активного экрана в процентах
Window& Locate( X,Y, Width,Height )     // позиционирование окна по экрану
virtual bool KeyBoard( byte key ) // транзакция по запросу Key, с возвратом
Window& KeyBoard( bool(*inKey )(byte) )  // установка внешней обработки
byte GetKey(),ScanKey()   // выборка символа и опрос готовности без остановки
byte ScanStatus()      // контроль сопутствующих кодов от управляющих клавиш
byte WaitKey()                  // запрос с ожиданием ввода символа от клавиатуры
virtual Window& Timer()       // виртуальная процедура для прерываний таймера
Window& SetTimer( mSec, bool( *inTime )()=null )  // время и транзакция
Window& KillTimer()                                  // сброс таймера
int isTimer // уровень рекурсии транзакций по таймеру для данного графического окна
Window& Refresh()  // последовательная перерисовка всех графических площадок по
        // признакам PlaceAbove с использованием ранее распределенной памяти Img
Window& Above()    // перемещение окна Window на верхний видимый уровень экрана
void    Help( Title,Cmd,Adj, x=-1,y=1 )  // три блока текстовых подсказок
              // Title – заголовок; Cmd – список команд; Adj – дополнения подсказок
```

:: *//! управление единым таймером вне окна и без задержки текущих вычислений*

```
DWORD WaitTime( DWORD Wait,// интервал задержки текущего потока вычислений
                bool(*Stay)()=null,  // внешний вычислительный эксперимент
                DWORD Work=0 )     // время на исполнение иных рабочих циклов
      GetTime()                  // текущее компьютерное время в миллисекундах
      ElapsedTime()         // время от момента запуска программы (~49,7 суток)
      StartTime      // отсчет времени по запуску исполнения активной программы
      RealTime  // практическое время исполнения процесса (inStay) внутри WaitTime
```

Place:: *//! основные графические операции на рабочей площадке OpenGL*

```
Place( Window*,mode )      // конструктор создания и привязки площадки к Window
       mode=PlaceOrtho   // масштаб с единичными кубом [-1:1], либо растр {w,h}
       mode=PlaceAbove// стековое наложение площадок над изображением, иначе
                      // сохранение графики при каждом проявлении буфера через Show
Place& Area( X,Y, Width,Height )        // определение размерений площадки
     //  X,Y > 0 – отсчеты от левого верхнего угла, <=0 – от правого нижнего
     //  Width,Height > 0 – отсчеты в символах, если =0 – до границы окна,
     //          если < 0 – в пикселях и естественных отсчетах Y – снизу вверх
```

```
Place& Activate( bool Act=false )      // активизация графического контекста
                Act=true // PlaceOrtho ? {w,h}:[-1:1] с запросом выбранного масштабирования
virtual Place& Mouse( x,y )            // движение в поле графической площадки
virtual Place& Mouse( state, x,y )     // реакция нажатия клавиши бышки
virtual Place& Draw()                  // виртуальная процедура обновления изображения
Place& Mouse( bool(*inPass)( int,int ) )          // внешняя обработка
Place& Mouse( bool(*inPush)( int,int,int ) )      // прерываний от мышки
Place& Draw( bool(*inDraw)() )         // отсылка к внешнему процессу отрисовки
Place& Clear( bool=true )              // очистка фоновым/true или текущим/false цветом
Place& Show()     // копирование графического фрагмента из активного буфера, с его
                  // пересохранением в связной памяти при наличии признака PlaceAbove
Place& Save() // безусловное сохранение текущего фрагмента изображения в связной
              // памяти, вне зависимости от (не)установки признака PlaceAbove
Place& Rest()         // восстановление фрагмента из связного списка в оперативной
                      // памяти в буфер OpenGL без проявления изображения на экране
Place& Refresh() // перерисовка всех наложенных площадок фонового окна Window

Place& Alfabet( h=0,Fnt="Courier New",weight=FW_NORMAL,italic=false )
Place& AlfaBit( Fnt=_8x08|_8x14|_8x16) // растровый шрифт из эпохи СССР
Place& AlfaVector( Real H=15, T=0 )    // векторный шрифт от Borland.chr
SIZE  AlfaRect( char *Text )           // размерения текстовой строчки в пикселях
Place& Print( x,y, Fmt, ... )          // лист сверху/слева, y/x<=0 – снизу/справа
Place& Print( Fmt, ... )               // контекстная печать по поверхности окна
Place& Text( Course,X,Y,Z, Fmt,... )   // текст в графическом контексте
Place& Text( Course, const Real *P, Fmt, ... )  // активных координат
extern byte _8x08[],_8x14[],_8x16[]    // ссылки на растровые шрифты
```

:: ***//! контроль и предустановка контекста для прорисовки графики и текста***

```
Window* Place::Ready() // запрос активности или текущего адреса для связного окна
bool WinReady( Window*=null ) // такой же запрос по окну или всей среде Window
bool glAct( Window* )  // явная привязка окна к графического контексту Window
class glContext( Window* )      // временное сохранение среды Window-OpenGL
                        // constructor ≈ (пролог) контекстного графического конвейера
                        // destrucor  ≈ (эпилог) – восстановление былого 3D-контекста
class RasterSector( X,Y,W,H ) // сектор растровых манипуляций под glViewPort
class TextContext( false )  // пролог текстовых записей, true – плюс базисы в стек
```

Window:: *//! окошко для кратких подсказок с управляющими кодами программы*

`void` **`Help`**`( char *Name[],char *Text[],char *Plus[],X=-1,Y=1 );`

Name[0] = *подзаголовок – название набора инструкций для всего Window*

Name[1-3] *– три строки расширенного названия (–и признак движения)*

Text – парное описание команд и операций с кратким предназначением

Plus – то же для блока дополнительных инструкций и подсказок

++ *завершение каждого блока со строчками заканчивается нулевым адресом*

`::` ***//! текстовое меню перенастройки и управления вычислительным экспериментом***

`struct` **`Mlist`**`{ short skip,lf; const char *Msg; void *dat; };`

skip – пропуск строк при построении меню запросов,

lf – длина поля текстового и числового запроса, или длина строчки-команды

Msg – сообщение с форматом запроса: % °|:, [l]defg, s ← *концевой символ*

dat – адрес числовых или текстовых данных для изменения.

`class` **`TextMenu`**`( Mlist,L,Window*,x=1,y=1 )` *// текстовое меню команд и запросов*

Пакет диалога с терминалом с помощью меню текстовых таблиц запросов

Mlist – список параметров для запросов на терминал

Num – количество записей с запросами в списке Mlist

Y, X – координаты левого верхнего угла для окна запросов

return – номер последнего активного запроса

`void` **`Break`**`( char Msg[],... )` *// для завершения, и если *Msg='~' – информация*

О транзакциях на прерываниях от исполнительной среды Windows

Как для виртуальных, так и для всех свободных транзакций, в момент прерывания происходит связывание окна Window с контекстной графической средой OpenGL с помощью Place::Activate(), затем (частично) запоминается текущее состояние и перенастраивается новая исполнительная среда для исполнения свободной транзакции. По выходу из прерывания средствами Window::Place происходит автоматическое восстановление исходной исполнительной среды, и продолжаются прерванные вычислительные и графические процессы.

Если процедура обработки прерываний возвращает *false*, то в Window::Place никаких дополнительных действий по визуализации не производится, что важно для организации эффективных параллельных (реентерабельных) математических вычислений в режиме прерываний, что частично снимает особые сложности в поддержании единого контекстного потока графического конвейера.

```
Place::          // виртуальные и свободные транзакции, связанные с площадкой Place
bool( *extDraw )()      // синхронная отрисовка картинки, при true – визуализация
bool( *extPass )( int X,int Y )      // две свободные процедуры обработки
bool( *extPush )( int State,int X,int Y )    // прерываний курсора мыши
```

Draw()→true – виртуальная транзакция прерывания WM_PAINT реагирует выводом стека изображений **Refresh**(). *В других случаях* **Draw**() *может вызываться* *только явно*. В базовой **Draw**() может исполняться **extDraw()→true**, с реакцией в Window:: через **Save().Refresh**(), и на площадке Place:: только **Show**().

Аналогичная реакция виртуальных **Mouse**(x.y) и **Mouse**(b,x,y), в которых связь с верхней по стеку площадкой Place:: реально отслеживается.

```
Window::         // прерывания таймера и отклики на клавиатуру основного окна OpenGL
bool( *extKey )( byte );  // процедура обработки прерываний от клавиатуры
bool( *extTime )()       // свободная транзакция прерываний от таймера Window
bool( *extFree )()      // и процедура в исполнительном цикле таймера программы
```

KeyBoard(*key*) и **extKey**(*key*) → **true** – если *key* принят, и **false** – отвергнут. Не принятые в прерываниях символы обслуживаются в очереди ожидания **WaitKey**() или в циклах опросов: **GetKey**(), **ScanKey**() и **ScanStatus**().

Timer() и **extTime**() → **true** работают в предустановленной среде OpenGL, и положительно реагируют сохранением и визуализацией всего стека окна Window:: с помощью **Save().Refresh**().

Независимый таймер DWORD **WaitTime**(*Wait,inFree(),Work*) по аналогии с вызовом **Sleep**(*mSec*) приостанавливает исполнение текущего потока на время *Wait* [мСек], но при этом сохраняется активность всех других процессов и прерываний в вызывающей программе **Windows**.

Вторым параметром указывается ссылка на свободную процедуру bool **extFree**() → *true*, которая циклически исполняется внутри **WaitTime** в течение указанного третьим параметром интервала времени *Work* [мСек], после чего заново приостанавливается всех процессов по кванту времени *Wait* – в течение которого возможно проведение служебных операций и прерываний от OS-Windows, необходимых для визуализации результатов и интерактивного управления вычислительными процессами.

Независимый вычислительный процесс может быть прерван при возврате свободной функцией **extFree**() → *false*. На выходе **WaitTime**(*Wait,inFree(),Work*) предоставляет практическое время, потраченное транзакцией **extFree**().

Внешнее обрамление типовых операций Window-Place (*окружение* Type.h *и* View.h)

Два файла описаний сосредотачивают множество привычных констант, базовых функций и файловых операций ввода-вывода в программировании для ОС-**Windows** на чистом **C++** в подборке «**Type.h**», и дополнительно для примитивов **Window-OpenGL** в файле исходных описаний «**View.h**», отчасти покрывающих базовые графические процедуры и виртуальные запросы в реализациях прямых вычислительных экспериментов.

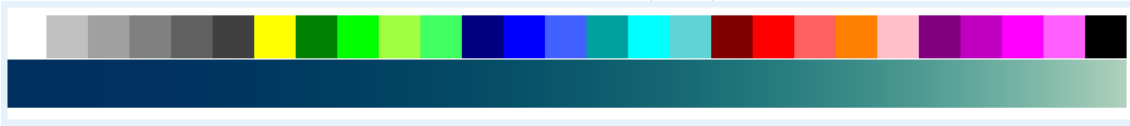
Type.h *// часто используемые общепрограммные константы и операции*

```
  a 6 378 245 м Сфероид Красовского для морских карт России
  b 6 356 863,0188 SN\φ                                  — эллипсоид Красовского
  c 6 399 698,9018 EW/λ (b+c)/2 = 6 367 554.0094   ε≈298.3
  м 1 855,35626248109543 м                               — сфероидальная миля
  ρ ≈ 1,025 кг/дм³ ≡ 25‰          — плотность морской воды (‰ – промили)
  c ≈ 299 792 458 ±1,2 м·с⁻¹      — скорость света в задачах электродинамики
_Mile=1 852,24637937659918      — морская(равнообъёмная) миля – 1' меридиана
      // радиус эквивалентной сферы в отношении к равнообъёмному единичному кубу
EqSphere=0.62035049089940001666800681204778     — r = ³√( 3 / 4 / π )
————————=1.24070098179880003333601362409556     — D = ³√ ( 6 / π )
  _Pi=3.14159265358979323846264338327950288     — π
  _Pd=6.28318530717958647692528676655900576     — π × 2
  _Ph=1.57079632679489661923132169163975144     — π / 2
  _iP=0.31830988618379067153776752674503        — 1 / π
  _Rd=57.29577951308232087679815481 4105        — 180 / π   °\ rad
  _dR=0.01745329251994329576923690768489        — π / 180   rad \°
  _e = 2.71828182845904523536028747135266249
  _g ≈ 9,8106 м/с² ≈ 9.780318·(1+0.005302·sin²φ-0.000006·sin²2φ)-0.000003086·h
   φ ≈ 1.61803398874989484820458683436563811 = 1/φ+1=(√5+1)/2
enum Course
{ _North_West=3, _North=1,_North_East=9, _Home=3,  _Up =1, _PgUp=9,
        _West=2, _Zenith=0,     _East=8, _Left=2,_Center=0,_Right=8,
  _South_West=6, _South=4,_South_East=12,_End =6, _Down =4,_PgDn=12,
  _Enter=13,_BkSp,_F1,_F2,_F3,_F4,_F5,_F6,_F7,_F8,_F9,_F10,_F11,_F12,
  _Esc=27,_Ins,_Del,_Tab,_Blank=32 };                  // +5,+7,+10,+31 — в запасе
enum{ _MouseMove,_MouseLeft,_MouseRight,_MouseMiddle=4,_MouseWheel=8 }
enum{ RIGHT=1,LEFT,SHIFT,LCTRL,RCTRL=8,CTRL=12,L_ALT=16,R_ALT=32,ALT=48}
struct Event{ long D; Real T;          // Юлианская дата и время суток в часах
  Event& Now(); Event& UnPack( &m,&d,&y ); long Pack( m,d,y ); } // перерасчёты времени
class string{ char *str; int len;       // текстовая строчка неограниченной длины
                char& operator[]( int ); }   // управляется оператором выборки символа
char* fname ( const char* FileName ) ;  // выборка собственно самого имени файла и
char* sname ( char* ShortFileName ) ; // тоже имя, но с отсечённым расширением,
char* fext  ( char*, const char* Ext=0 ) ; //или с принудительной заменой FileName.ext
FILE *FileOpen ( char *fn, const char *tp, const char *ex, const char *ch, const char *tl );
char *getString ( FILE *F ) ;          // чтение строки на едином статическом адресе
char *getString ( FILE *F, int tab ) ;  // с неограниченной длиной; без(-)табуляторов

void *Allocate ( size_t Sz, void *A=NULL );              // Линейный массив в байтах
void **Allocate ( size_t Ny,size_t Sz,void A* =NULL ) ;  // Количество и длина строк
size_t isAlloc ( void* A ) ;  // ?оперативный объём или количество строк в матрице
```

View.h *// подборка основных констант и операций контекстной графики*

```
void View_initial()      // ! начальная инициализация графической среды OpenGL
const char                // словесные прописи имён месяцев года и дней недели
*_Mnt[]={"январь","февраль","март","апрель","май","июнь","июль","август","сентябрь",
*_Day[]={"понедельник","вторник","среда","четверг","пятница","суббота","воскресенье"};
enum colors{ white,silver,lightgray,gray,dimgray,darkgray ,freeboard,
  yellow,green,lime,olive,lightgreen,navy,blue,lightblue,cyan,aqua,
  lightcyan,maroon,red,lightred,orange,pink,purple,magenta,fuchsia,
  lightmagenta,black,empty = –1   +SeaColor( 256 )  }  // != 28\{29}
```



```
color( colors clr )  // выбор одного цвета, дополняемого палитрой SeaColor+256
color( colors clr,     // ... подстройка с относительной подсветкой / затенением
           bright,              // ... от белого <= +1,0  # –1,0 => до чёрного ...
           alfa=1 )          // прозрачность \ смешивание – 1 => 0 – выцветание
#define aR const Real*       // доступ к вектору(Vector) и точке(Point) по ссылке
aR dot ( aR a ){ glVertex3dv( a ); return a; }      // контекстная точка так, как есть
aR dot ( aR,colors )                  // та же точка с предустановкой цвета
aR spot( aR,Size,colors=empty )       // рисунок • точек с размером и цветом
aR line( aR,aR )                      // завершённый отрезок — прямой линии
aR line( aR,aR,colors )               // та же линия с предустановкой цвета
void liney( aR,aR,colors=empty )      // та же линия с дублем по ординате y
void rectangle( aR LD, aR RU, bool=true )  // прямоугольник плоскости {x-y}
aR circle( aR center, radius, bool=true )  // круг или окружность на {x-y}
aR arrow( aR a,aR b,ab=0.06,colors=empty )  // линия со стрелкой ab на конце
                  // разметка координатных осей с чуть затемнёнными надписями xyz
void axis( Place&, X,Y,Z, «ось X»,«ось Y»,«ось Z», colors=cyan )
View:Window:   //! фоновые виртуальные операции с графическим окном в OpenGL
View( Title,X,Y,W,H, Size=1 ) // новое окно Window и начальные предустановки
virtual bool Draw() // перестраивается графическая сцена или новое изображение
virtual bool Mouse( x,y )       // отслеживаются текущие координаты мыши
virtual bool Mouse( state,x,y ) // здесь сдвиги и повороты графической сцены
virtual bool Keyboard( key )    // те же сдвиги-повороты в окне с клавиатуры
```

В алгоритмах виртуальных процедур **View**:Window включается интерактивный пролог воздействия на графическое изображением, с последующим наследованием:

Draw – настраивает графическую сцену в перспективной проекции по дистанции и ориентации, с последующей подстройкой направления её внешнего обзора.

Mouse – с нажатой левой кнопкой выполняется вращение сцены относительно её действующего центра; с правой кнопкой происходит простое смещение сцены; вращением колёсика изображение удаляется или приближается. Если задействуется клавиша <Ctrl>, то левая кнопка мыши будет наклонять изображение при движении вправо-влево, и работать вместо колесика для изменения дистанции при движении вверх и вниз.

Keyboard – <Home> - приводит изображение к изначальному состоянию; стрелки <◄▲▼►> – вращают сцену подобно мышке, или смещают её в аккорде с клавишей <Shift>; и с <Ctrl> – аналогично наклоняют и меняют дистанцию до графической сцены.

Window – Place

Обобщенные структуры объектов и операций трёхмерной графики OpenGL с контекстным интерфейсом виртуальных функций в C++

Предварительные краткие наставления

Объявление базового или производного объекта **`Window`** создает на графическом экране новое окно с полноразмерной фоновой площадкой **`Place`**, что задействует основные операции ***`OpenGL`*** и периферию компьютера. В конструкторе **`Window`** заголовок *Title*, местоположение *X,Y* и размеры окна *W,H* в пикселях экрана: (+) от левого верхнего и (–) – от правого нижнего угла. Без заголовка – окно фиксированного размера без рамок (перемещается мышкой с клавишей <Alt>). Положение и размеры окна можно изменять процедурой Window::**`Locate`**(*X,Y,W,H*), где параметры могут задаваться в процентах от экрана с помощью макросов Xpm(*X*) и Ypm(*Y*).

Графическая площадка **`Place`** объявляется со ссылкой на активное окно **`Window`**, вторым параметром конструктора признаки: *Signs*=`PlaceAbove` для наложения площадки с контролем фонового изображения, и `PlaceOrtho` – размечает масштаб на вмещение куба с размерностями [–1 : 1], при отсутствии – размерности в пикселях [0÷*w*, 0÷*h*, –1÷1]. Местоположение и размеры площадки внутри окна определяются процедурой Place::**`Area`**(*x,y, w,h*), где положительные величины задают количество символов и строк сверху-слева; отрицательные – в точках растра для отстояния от границ. Выполнение Place::**`Activate`**(*act*) активирует операции **`OpenGL`** на заданный фрагмент, где параметр act=true – означает предустановку масштабирования по условию `PlaceAbove`.

В ***`OpenGL`*** изображение формируется в буфере, и Place::**`Show`**() делает его видимым. Без признака `PlaceAbove` не тратятся ресурсы на частый сброс графики в связную оперативную память, полагая возможность принудительного сохранения с помощью Place::**`Save`**() после формирования изображения. Возврат графики из связной памяти в активный буфер изображения выполняет Place::**`Rest`**(). Полное обновление окна с последовательным переналожением всех площадок: Window::**`Refresh`**().

Доступны растровые: **`AlfaBit`**({_8x08,_8x14,_8x16}) и TrueType шрифты Windows: **`Alfabet`**(*y*=0, "Courier New", *weight*=FW_NORMAL, *italic*=false). Процедуры: Place& **`Print`**(*fmt, …*) + **`Print`**(*x,y, fmt, …*) – построчная печать текста как по листу, и **`Text`**(*Dir,x,y,z, fmt, . . .*) – простые надписи с пространственной 3D привязкой.

Четыре функции опроса клавиатуры: Window::**`WaitKey`**`()` + **`GetKey`**`()` + **`ScanKey`**`()` + **`ScanStatus`**`()`, с ожиданием, считыванием и запросом наличия символа или управляющего кода: Alt, Ctrl, Shift.

Запрос адреса активного окна Window* Place::**`Ready`**() приводит к ожиданию исполнения всех операций в очередях Windows, а аналогичная функция bool **`WinReady`**(Window*=*null*) также проверяет активность заданного Window, либо наличия первого в списке существующих в программе окон.

Построение вычислительных процессов регулируется независимой процедурой **`WaitTime`**(*Wait*, bool(**inFree*)()=0, *Work*=0), ожидающей *Wait* [мСек] исполнения внутренних циклов системы, или организующей вызов внешней транзакции *extFree()* с циклически ограничиваемым по времени *Work* [мСек] ресурсом.

Включены в работу виртуальные транзакции с передачей управления по наследованию: Place::**`Draw`**() + **`Mouse`**(*x, y*) + **`Mouse`**(*b, x, y*); и Window::**`KeyBoard`**(*key*) + **`Timer`**(); и они же свободные: **`Draw`**(bool(**inDraw*)()) + **`Mouse`**(bool(*)(*x,y*)) + **`Mouse`**(bool(*)(*b, x, y*)); **`KeyBoard`**(bool(*)(*key*)) и **`SetTimer`**(*mSec*, bool(*)()=0).

Контекстная систематизация графических операций

Графическая среда OpenGL изначально построена на контекстно-зависимых операциях, что обусловливалось относительно медленным однопоточным каналом связи между собственно вычислительным ядром и независимой графической станцией. Контекстная зависимость графических операций затрудняет параллельное исполнение реентерабельных (повторновходимых) процедур, и, как следствие, не допускает многопроцессорного распараллеливания, с требованием особого визуального контроля реакций на прерывания незавершенных последовательностей графического конвейера OpenGL.

С учётом построения графической среды разделяются процедуры Window::Place:

– по исполнению в контекстной привязке к активному графическому окну;

– процедуры с выбором и переназначением графического контекста;

– особые транзакции для исполнения прерываний с предустановкой и быстрым восстановлением графической среды незавершенных алгоритмов.

1) операции с фиксацией контекста OpenGL в окне Window формально служат базисом для последовательностей контекстно зависимых графических операций:

– **`Window`** и **`Place`** конструкторы всегда оставляют связь с контекстом OpenGL.

– bool::**`glAct`**(Window*) – явная привязка окна к графического контексту;

– Place::**`Activate`**(mode=*false*) – предустановка Place в связанном окне Window с графическим контекстом OpenGL, где mode=*true* к выбору масштаба по признаку `PlaceOrtho` – единичного куба, либо – растрового листа { *w,h* }.

– Window::**`Locate`**(x,y, w,h) – изменение размеров активного окна Window;

2) подборка процедур для изображений с привязкой к любому окну Window при сохранении текущей активности графического контекста OpenGL:

– Place::**`Area`**(*x,y, w,h*) – назначение места и размерений графической площадки;

– Place::**`Alfabet`** и **`AlfaBit`** – установка шрифта для графического фрагмента;

– Place::**`AlfaRect`**, **`String`**, **`Text`**, **`Print`** – формирование и пропись текстов;

– Place::**`Clear`**, **`Save`**, **`Rest`**, **`Show`** – операции с видимым изображением;

– class **`RasterSector`**(*x,y,w,h*) и **`TextContent`**(*Space*) – пролог с последующим эпилогом подстройки контекста для растровых и текстовых фрагментов;

– все процедуры пакета View: virtual **`Draw`**, **`Mouse`**, **`Keyboard`**, и независимые: **`View_initial`**, **`axis`**, **`arrow`**, **`point`**, **`line`**, **`color`** и др. …

– class **`RasterSector`**(*x,y,w,h*) и **`TextContent`**(*Space*) – пролог с последующим эпилогом подстройки контекста для растровых и текстовых фрагментов;

– все запросы к клавиатуре **`WaitKey`**, **`GetKey`**, **`ScanKey`** и **`ScanStaus`**, также как и к таймеру **`WaitTime`**, **`SetTimer`** и **`KillTimer`** связаны только с очередями и прерываниями окна Window, и при этом не выполняется ассоциирование с графическим контекстом OpenGL.

– Window::**`Ready`** и **`WinReady`**(Window*) – среды OpenGL никак не касаются.

3) процедуры с временным задействованием графического контекста OpenGL предназначены, в первую очередь, для корректной работы в условиях прерываний:

– class **`glContext`**(Window*) – конструктор как пролог, деструктор – эпилог.

– Window::**`Refresh`** – обновление всех площадок не фиксирует связь с OpenGL.

– Все виртуальные транзакции и аналогичные процедуры обработки прерываний на входе получают предустановленным графический интерфейс в OpenGL, который неявно возвращается к исходному по завершению прерывания.
Это те же: virtual **`Draw`**, **`Timer`**, **`KeyBoard`** и **`Mouse`** и свободные аналоги.

Производный класс Window на базе Place

Оконный интерфейс Window для OpenGL в среде Microsoft Windows.

```
class Window: Place                    // стандартное окно Windows для OpenGL
```

Производный класс Window открыто наследует элементы базового класса – исходной графической площадки Place, и замыкает на себя комплекс операций для доступа к внешней периферии: графическому экрану и клавиатуре, с поддержкой операций для проведения вычислительных экспериментов и и визуализации результатов под управлением интервального таймера.

Конструктор Window создает элемент списка графических окон с опорным статическим адресом Window* **First**. Завершающий элемент списка имеет нулевую ссылку Window***Next**, как бы для отсутствующего окна.

В каждом окне фиксируется нижний элемент стека налагаемых площадок в базовом классе: Window*Place::Site = Window::this. Активность Window поверяется процедурами Window::Ready и WinReady(Win), в которых сначала исполняется внутренняя очередь операций Windows, затем сверятся наличие адреса Site, который может быть обнулён деструктором базовой площадки.

Информацию о размерностях графического экрана на момент создания нового окна Windows хранится во внутренних константах структуры Window:

```
int ScreenWidth, ScreenHeight          // полные размеры экрана ЭВМ
```

Для позиционирования относительно этих размеров в процентах (%%) от всего активного графического экрана, предусмотрены макросы с обращениями к функциям Win32:

```
#define Xpm( X ) ( GetSystemMetrics( SM_CXSCREEN ) * Real( X )/100.0 ) // %%X
#define Ypm( Y ) ( GetSystemMetrics( SM_CYSCREEN ) * Real( Y )/100.0 ) // %%Y
```

В неявном конструкторе класса Window использованы следующие значения по умолчанию:

```
Window::Window( char* Title=NULL,
                int X=0,int Y=0, int Width=0,int Height=0 )
```

что определяет простое окно без рамки с графическим полем 800x600;

Если указан заголовок **Title**, то создается стандартное окно Windows с активной рамкой с верхним заголовком и управляющими кнопками.

Если заголовка нет (**Title**=*null*), то создается простое окно заданного размера без активной рамки. Размеры такого окна невозможно изменить извне, что не снимает необходимости контроля и перерисовки изображения по внешним прерываниям.

Числовые параметры **X**, **Y**, **Width** и **Height** определят местоположение и размеры полного графического поля внутри Window.

X, **Y** – положительные величины определяют местоположение левого верхнего угла { 1,1 } нового окна Window, отрицательные величины – задают соответствующие отступы от правой и нижней границы графического экрана ЭВМ. Нулевые значения **X**, **Y** – ставят окно на четверть отступа сверху и треть – справа.

Width и **Height** – ширина и высота выделяемого окна Windows. Нулевые значения заменяются величинами 800x600 – соответственно; отрицательные или слишком большие значения приводят к установке максимальных размерностей окна в пределах всего графического экрана ЭВМ. Обрамляющие рамки Windows добавляются к исходным размерам Width и Height.

Для динамического изменения размеров и местоположения окна Window предназначена процедура **Locate**, числовые параметры **X**, **Y**, **Width** и **Height** интерпретируются также, как и в вышеописанном конструкторе:

```
void Window::Locate( int X, int Y, int Width, int Height  );
```

Отсчеты местоположения и размеров окна могут быть заданы в процентах относительно экрана ЭВМ с помощью функций – макросов: **Xpm**(X) и **Ypm**(Y). При определении реальных параметров окна, по необходимости смещаются контрольные отсчеты местоположения – **X**, **Y** в пользу поддержания максимально допустимых величин – **Width** и **Height**.

Подборка основных процедур для работы с клавиатурой

Шесть процедур для получения данных в программе с помощью клавиатуры всегда связаны с конкретным окном **Window**, и все посылки с клавиатуры сохраняются в его кольцевом буфере до момента выборки внутри в программы:

```
byte WaitKey()                  // остановка и ожидание нового символа с клавиатуры
byte GetKey()                   // запрос и выборка символа без остановки программы
byte ScanKey()                  // опрос символа без остановки и без выборки из очереди
byte ScanStatus()               // получение из буфера кода для сопутствующих клавиш
virtual bool KeyBoard( byte )   // виртуальная процедура по умолчанию вызывает
Window& KeyBoard( bool( *inKey )( byte ) )      // регистрация свободного модуля
    // обработки прерываний для реагирования на ввод команд и данных с клавиатуры
```

WaitKey и **GetKey** выбирают по одному символу из буфера, **ScanKey** показывает его поступление в буфер. Если окно Window закрывается извне, например <***alt-F4***>, то возвращается 0, и также обнуляются Window* **Site** и запрос к Place::**Ready**().

Функция **ScanStatus** считывает признаки сопутствующих <***Shift***>, <***Alt***> и <***Ctrl***> клавиш, нажатых сейчас или ранее в момент успешного ввода символа в буфер клавиатуры, и могут принимать следующие значения/маски:

```
RIGHT=1,   LEFT=2,    SHIFT=3,    // 0x03
LCTRL=4,   RCTRL=8,   CTRL=12,    // 0x0C
L_ALT=16,  R_ALT=32,  ALT=48.     // 0x30
```

<***Ctrl***+C> – нормальное завершение с исполнением всех деструкторов;
<***Alt***+*LeftMouse-move*> перемещение окна по экрану ЭВМ.

Виртуальная **KeyBoard**(*key*) и свободная **extKey**(*key*) получают один символ с клавиатуры, и возвращают *true* для продвижения к следующей ячейке указателя клавиатурного буфера из 64 позиций. Если символ не принят в работу, функции обработки прерываний возвращают *false* – создавая, тем самым, опасность блокировки ввода с клавиатуры.

При остановке программы по **WaitKey**() происходит отключение обработчиков прерываний от клавиатуры, что требуется для временного захвата клавиатуры, например для ввода текстовых или числовых данных, или для позиционирования курсора в строчках текстового меню

Комплекс процедур интервального таймера

Предусматривается один интервальный таймер для программы в целом, и с каждым окном **Window** может быть связан собственный виртуальный таймер, которые полу-

чают управление по заданному интервалу времени в последовательности выборки прерываний на исполнение внутренних очередей программы в Windows.

Общий таймер для управления вычислительным экспериментом:

```
DWORD WaitTime( DWORD Wait,        // активная задержка для независимых операций
      bool(*inFree)()=0,  // свободная функция цикла вычислительного эксперимента
      DWORD Work=0 );  // контрольное время на исполнение цикла вычислений [мСек]
```

По умолчанию данный таймер вводит программу в цикл исполнения операций из внутренней очереди Windows с опросами WaitMessage на время *Wait* [мСек], что обеспечивает корректную работу с внешними устройствами и графическим экраном. Если указывается ссылка на свободную вычислительную процедуру bool *extFree*(), то внутри **WaitTime** организуется непрерывный цикл на время *Work*, для управляющих запросов и графического представления результатов в течение *Wait*.

Для выхода из цикла, процедура *extFree()* должна вернуть значение *false*. Можно прекратить внутренний цикл повторным обращением **WaitTime**(0), если такое возможно в свободном от управляющих связей вычислительном модуле *extFree*(). **WaitTime** – возвращает чистое суммарное время, потраченное на вычислительный эксперимент в цикле исполнения модуля *extFree*().

С окном Window связаны три программы для работы с таймером, включая виртуальную процедуру **Timer**, для которой производится фоновая предустановка, настройка и масштабирование контекстной среды OpenGL.

```
Window& SetTimer( DWORD mSec, bool(*inTime)() ) // интервал и свободная процедура
virtual bool Timer()        // виртуальный модуль обработки прерываний таймера
Window& KillTimer()              // сброс таймера — установка нулевого интервала
```

Если виртуальная процедура не перекрывается в производных классах, то с базовыми предустановками может быть вызывана свободная транзакция bool *extTime*(), задаваемая вторым параметром в Window::**SetTimer**(*mSec*, bool(*inTime)()).

Если заданный интервал *mSec* меньше реального времени исполнения процедуры обработки прерывания, то последующие виртуальные **Timer** или свободные *extTime* транзакции будут игнорироваться до завершения уже работающей.

Если обработчики прерываний **Timer** или *extTime* возвращают *false*, то каких-либо действий по визуализации результатов не требуется. В случае *true* – контекстная графика переносится сначала в связанный буфер с помощью **Save**(), с последующим восстановлением и визуализацией итогового изображения по **Refresh**().

Следующие глобальные переменные и функции дают доступ к использованию компьютерных часов:

```
DWORD StartTime, // время запуска исполнения программы от начала работы Windows
      RealTime;        //  время исполнения параллельной процедуры внутри WaitTime
DWORD GetTime(),       // текущее время в миллисекундах = timeGetTime = GetTickCount
      ElapsedTime(); //! время работы программы, опрокидывание через ~49,7 суток
```

Скрытые (private:) внутренние ссылки на процедуры обработки прерываний в контексте окна Window:

```
bool (*extKey)( byte ); // свободная процедура обработки прерываний клавиатуры и
bool (*extTime)();    // внешняя процедура для отработки срабатывания таймеров,
                      //  на выходах запросы на обновление изображении всего окна
```

Базовый класс: Place – контекстная графическая и текстовая среда наложенных страниц

Наложение графических фрагментов Place по поверхности окна Window

`class` **`Place`** // графическая площадка/страница на поверхности окна Window

Основной графический объект, обеспечивающий контекстную графику и стандартные текстовые запросы средствами **OpenGL** на специально выделенных фрагментах в поле **Window**, размечаемых наложенными поверх прямоугольными площадками как: class **Place**. Одновременно **Place** является базовым классом для всего окна **Window**, что требуется как в качестве шаблона для всех наложенных фрагментов, а также в случае вовлечения всего графического окна в качестве простейшего или единственного графического интерфейса **Window::Place**.

С базовым классом связываются все контекстные операции OpenGL, а также системно-зависимые утилиты для позиционирования и сохранения растровых полей; выбора шрифтов и представления текстовых строк в графическом и страничном форматах; обработки прерываний от указателя «мышь» и др.

`Place::`**`Place`**(byte *Signs* = `PlaceOrtho | PlaceAbove`) // конструктор

```
struct Window;                  // родительский класс определяет рабочее окно Windows
struct Place                    // базовый класс графической площадки/текстового листа
{ Window *Site;                   // опорный (для Place) контекст окна Window в Windows
  byte Signs;                     // особые режимы/признаки управления страницей Place
  Place *Up;                      // адрес в последовательном списке наложений Window
  int  *Img;                      // временное хранилище фонового графического образа
  int pX,pY,Width,Height;         // положение и размеры на родительском окне Window
 struct hFont;                    // шрифт сохранятся подключенным к hDC Windows
 { byte *Bit;                     // временная установка старого растра из DispCCCP
   HFONT hF;                      // шрифт сохраняется для внутрисистемной метрики
   int Base,W,H;                  // индекс TrueType-OpenGL, ширина и высота символа
 } *Fnt;                          // ссылка на новый шрифт или базовый шрифт Window
 friend class Window;             // взаимный доступ к элементам связных объектов
  bool( *extPass )( int X, int Y ); // три адреса внешних независимых процедур
  bool( *extPush )( int State, int X, int Y );  // обработки прерываний от мышки
  bool( *extDraw )(); // Рисование по стандартному полю графического фрагмента (окна)
}
```

Конструктор новой площадки **Place** создает чистую заготовку, предварительно связанную с исходным контекстом структуры Window::**First**, что необходимо для доступа к контексту внутренних или исходных настроек графической площадки. В качестве обязательного параметра при конструкторе указывается маска битов для установки режимов использования новой наложенной площадки:

```
Enum Place_Signs
    { PlaceAbove=0x80,   // сохранение-восстановление изображения
      PlaceOrtho=0x40  } // масштаб на  трёхмерное ортогональное пространство
```

Бит **`PlaceAbove`** = 0x80 указывает на необходимость включения алгоритмов автоматического контроля и восстановления изображения графического фрагмента **Place** при обращении к визуализации Place::**`Show`**(). Аналогичное восстановление изображения будет выполняться принудительно после вызова операции Place::**`Save`**(), однако в этом случае режим автоматического обновления задействоваться не будет.

Бит **PlaceOrtho** = 0x40 включает представление пространственного куба с граничными размерами: `X[-1:1]; Y[-1:1]; Z[-1:1]`. Если бит **PlaceOrtho** отключён, то в качестве физических границ размечается поверхность в плоскости { *X, Y*} с растровыми размерениями [0, 0, *Width, Height*], где отсчеты координат ведутся из левого–нижнего угла. Такой естественный режим масштабирования удобен для работы с текстами, для которых известны растровые размеры шрифтов (Fnt.*Width*, Fnt.*Height*), или вычисляются размерения печатаемых строк по `SIZE` **AlfaRect**(*str ,bool=ANSI*): { long *cx,cy* }.

Выбранное масштабирование задействуется в случае указания значения *true* в параметре процедуры Place::**Activate**(*bool=true*), иначе, при указании значения *false*, выполняется только привязка исполнительной среды OpenGL и предустановка прямоугольного фрагмента `glViewport`(*pX,pY,Width,Height*), где координаты *pX, pY* – задают местоположение внутри окна Window.

Управление контекстной графической средой Place

Наложенные графические площадки **Place** обеспечивают полный и независимый интерфейс для управления фрагментами растрового поля, обеспечивающие привычную среду представления контекстной графики для **OpenGL**. То есть, виртуальная работа с графическим операциями в непосредственной связке с ОС-**Windows** может выполняться без уточнения принадлежности к окну **Window**, как бы с отсрочкой до выдачи специальных команд визуализации результатов.

С каждым фрагментом `Place` предопределяется конкретный растровый **AlfaBit** или стандартный **Alfabet**-(truetype) шрифт. Если привязка конкретного шрифта отсутствует, то тип шрифта будет выбираться из базового окна **Window**: `Place`, где по умолчанию предустанавливается немного утолщенный моноширинный шрифт «Courier».

По размерам активированного шрифта выполняется позиционирование и определение размеров наложенной площадки в **Area** (*pX,pY,Width,Height*) в отсчетах количества символов по **Alfabet**() или **AlfaBit**(), иначе такие размерения будут отсчитываться в точках графического растра.

`void Place::`**Area**(*pX,pY,Width,Height*) *// местоположение и размерности*

Установка местоположения и размеров наложенного в окне **Window** графического фрагмента. В процедуре **Area** не выполняются привязки контекста **OpenGL** к конкретному окну **Window**, что иногда полезно для прорисовки или копирования схожих графических площадок в разных окнах OS-**Windows**. С этой целью предусматривается установка размерений либо в абсолютных величинах, либо в отступах относительно границ окна, с указанием числовых размерений либо количеством текстовых символов, либо числами пикселей графического экрана.

Так, если *pX, pY* > 0 – отсчеты местоположения выполняются от левого и верхнего угла **Window**, иначе – для нулевых отрицательных значений *pX, pY* ≥ 0 от правого и нижнего, соответственно.

Width, Height > 0 – размеры площадки устанавливаются в количестве символов предустановленного шрифта от левой и верхней границ окна **Window**. Если *Width* = 0 или *Height* = 0 то границы площадки вытягиваются до противоположной границы окна, в предположении что величина *pY* теперь отмеряется от нижнего края в сторону верхнего.

Если *Width, Height* < 0 – размеры площадки определяются только в растровых отсчётах, с установкой правой системы геометрических координат, ось Y – снизу вверх.

Если шрифт предварительно не устанавливался, то размеры площадки **Area**(x,y,w,h) определятся по аналогичному алгоритму, как бы с размерами шрифта – в одну точку [1x1].

Если площадка создавалась с указанием режима **PlaceAbove**, то в процедуре **Area** выполняется пр200

едварительное сохранение фонового изображения, для поддержания возможности его быстрого растрового восстановления без запросов перерисовки по транзакции **Draw**().

Связывание наложенной графики Place с окном Window

Активность площадки `Place` и существование опорного `Window` проверяется вызовом функции `Window* Place::`**Ready**(), которая возвращает адрес окна или *NULL*, если работа с запрашиваемым окном прекращена. Эта чисто информационная функция не влияет на состояние исполнительной среды **OpenGL**, и для её практического ассоциирования с Window может применяться простейшая и быстрая операция `bool` **glAct**(*Window**), с подтверждением *true* при успешном подключении.

Для работа с контекстными операциями **OpenGL** необходимо предварительно исполнить процедуру `Place::`**Activate**(*bool=false*), в которой выполняется предустановка геометрического масштабирования будущих графических примитивов и операций с ними.

Вызов **Activate**(*false*) не сопровождается предварительным масштабированием фрагмента **Place**, а только активирует привязку **Window** к контекстной графике **OpenGL** и выставляет размерения границ прямоугольной площадки как: glViewport(*pX,pY,Width,Height*);

В случае вызова **Activate**(*true*) выполняется масштабирование активизируемого графического объекта и его привязка к окну **Window**. Указание ортогональных координат: **PlaceOrtho** = x40 предусматривает установку внутри графического фрагмента единичного куба: `X[-1:1]; Y[-1:1]; Z[-1:1]`. Ось X направлена слева-направо, Y – снизу-вверх, Z – из экрана на наблюдателя. Это нейтральная разметка для единичной матрицы, к которой применимо простое и вполне адекватное перемасштабирование. Так вызов glOrtho(0,1, 0,1,-1,1) переключит масштаб на вмещение куба: `X[0:1]; Y[0:1]; Z[-1:1]`. Без бита **PlaceOrtho** масштабирование сводится к плоским растровым или буквенным размерениями с границами из параметров процедуры **Area**(*pX,pY, Width,Height*): `X[0:`*Width*`]; Y[0:`*Height*`]; Z[-1:1]` (ось Y направлена снизу-вверх), что удобно для работы с растровыми изображениями и текстами.

Управление графическими результатами с переносами в оперативную память, в графические буфера и на экран с текущим изображением в окне Window-OpenGL:

```
Place&  Clear( true )   // исходная расчистка ограниченной графической площадки
Place&  Save()          // сохранение текущего изображения в оперативной памяти
Place&  Rest()          // восстановление фрагмента экрана из памяти
Place&  Show()          // проявление на экране фрагмента из графического буфера
Window& Refresh()       // обновление Window с переналожением площадок Place
```

(+++) Временно закрытые процедуры управления графическими фрагментами (как пока не востребованными в настоящей практике):

```
Place&  Dive()          //  стековое погружение вглубь с перестроением наслоений
Place&  Rise(int)       // подъем из стека с возвратом на указанное число площадок
Place&  Free()          // принудительное освобождение площадки от окна процедуры
```

Транзакции обработки прерываний от указателя «мышь»

Два варианта прерываний от указателя «мышь» предусматривают передачу управления при свободном движении над конкретной площадкой, либо при движении с нажатыми кнопками (*простое реагирование на нажатие кнопок мыши пока не встроено*):

```
virtual void Place::MouseMove ( int X, int Y )        // свободное движение или с
virtual void Place::MousePress ( int But, int X,int Y ) // нажатой кнопкой Button
```

Обе процедуры включаются в работу только при условии предварительного подключения внешних процедур обработки прерываний:

```
void Place::MouseMove  ( void(*inPass)(          int X, int Y ) )
void Place::MousePress ( void(*inPush)( int But, int X, int Y ) )
```

При вызове внешних независимых процедур обработки прерываний от указателя мышь: inPass и inPush, происходит предварительное переключение окна Window, сохранение текущего графического контекста OpenGL, а ссылке Place::**Act** передается адрес контекста площадки под указателем «мышь». Собственно вызов утилиты масштабирования: Active() не выполняется. По завершении прерывания восстанавливается фокус активности первоначального окна Window с собственным графическим интерфейсом OpenGL, что, *как правило*, достаточно для безаварийного продолжения работы прерванных операций.

Подборка растровых и векторных шрифтов

Ввиду несовместимости стандартных шрифтов Windows с иными операционными системами, при неразрешимости странных проблем в OpneGL-1.1 на компьютерах с процессорами AMD, в последних версиях графической среды Windoe:Place проприетарные шрифты True-Type заменены подборкой простых русских векторных шрифтов, аналогичных по формату с Borland.chr и DesignCAD.vct.

Формально, независимые векторные и растровые шрифты необходимы для обеспечения совместимости программ на уровне исходных кодов в различных операционных системах с графической средой OpenGL. Как вариант, интерфейс Windows может заменяться ОС-совместимыми надстройками, что также успешно отработано с использованием известной оболочки GLFW.

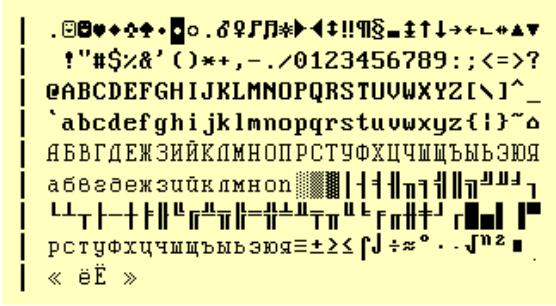
Для совместимости с иными операционными системами, как Linux\Unix; Andoid, OS-MESA и др., с состав среды программирования добавлена возможность замены прямого обращения к msWindows-WGL на GLFW-OpenGL. Переносимость текстовых шрифтов обеспечивается полностью автономным набором растров на Русском и Английском языках, а также комплектом векторных шрифтов со сдвоенными начертаниями контуров букв на Русском, Английском, Греческом и Иврите, с полным набором графических символов из старых OEM и ANSI кодировок.

```
Place& AlfaBit( DispCCCP )     // привязка растрового шрифта на 256 символов
 unsigned char _8x08[],_8x14[],_8x16[];  // просто русские растровые шрифты
```

В современных версиях Windows пропорции шрифтов выставляются настройками графических экранов, что вновь позволяет пользоваться старинными растровыми шрифтами **AlfaBit** с условно малыми размерами, так как приемлемые к прочтению размеры символов растеризуются автоматически (особенности текстов в окнах Windows).

В работу включены «старинные» шрифты из коллекции **DispCCCP** в трех вариантах: **_8x08**; **_8x14**; **_8x16** *(размерения в точках экрана)*, где русские буквы в исходной

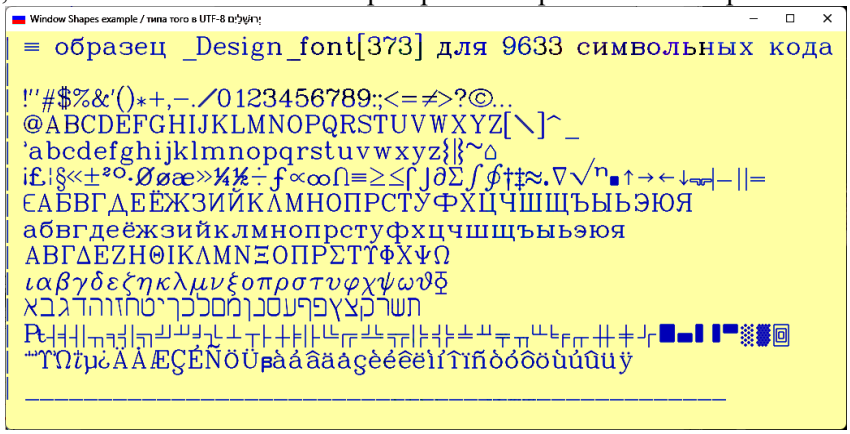
OEM-кодировке прорисованы тонкими линиями, а латинские — жирными. В текущей версии Window:Place повсеместно задействуется расширенная кодировка UTF-8, что позволяет дополнить растровые матрицы буквами **ё** и **Ё**, и символами русских угловых кавычек: **«** и **»**.



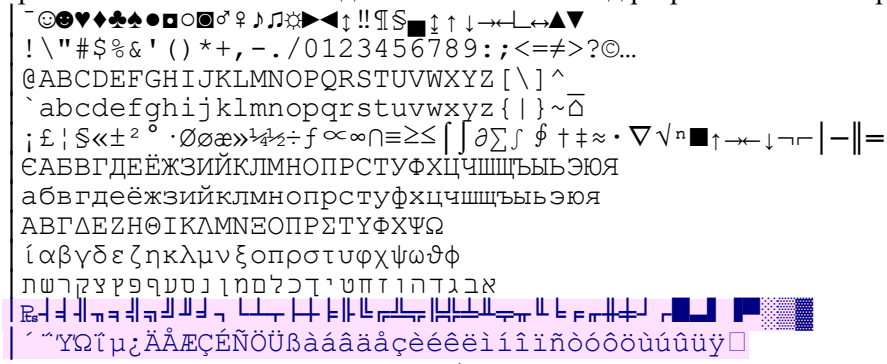
~~SIZE& Place::Alfabet(int=16, const char*="Courier", // установка TT-шрифта~~
~~byte weight=FW_DONTCARE, byte italic=false)// Windows~~

```
Place& AlfaVector( Real H=15, T=0 ) // векторный шрифт типа «Complex»
```

Текущий вариант графической прорисовки рабочих шрифтов показывается рисунком, схваченным захваченным с растрового экрана компьютера.



Две нижние выделенные фиолетовым фоном строки в рабочей версии обычно исключается, что экономит чуть более четырех килобайт в исполняемом коде программ. Для выборки. Исходный текстовый вариант шрифта приводится ниже, он может использоваться для копирования и вставки необходимых символов в код разрабатываемых программ:



Прямая прорисовка векторных шрифтов под контролем **AlfaVector** снимает ограничения по подборке символов из таблиц фиксированных размеров. Здесь вполне

уместно динамически размечать текстовые надписи, например в пропорциях от размерений графической площадки или изображаемого объекта, для чего все размерения шрифтов определяются числами в плавающем формате.

`SIZE Place::`**`AlfaRect`**`( char* )` *// растровые размерения надписи*

Процедура **`AlfaRect`** выдает размеры растрового представления строки, что может быт использовано, например, для предварительной расчистки.

void **`Print`**(int X, int Y, const char *_fmt, `...`) *// лист y/x<=0 — снизу/справа*
void **`Print`**(const char *_fmt, ...) *// контекстная печать*

Две процедуры позволяют печатать тестовые строки на графической площадке, как по писчему листу, с отсчетом первой позиции печатаемой строки от верхнего-левого угла при положительных X,Y, и от правого-нижнего при отрицательных X,Y, соответственно. В процедурах **`Print`** допускается многократное использование символа '`\n`' для перехода на новую строку.

`int` **`Text`** (Course Dir, Real X, Real Y, Real Z, const char* _fmt, ...) *// подписи в* 3D
`int` **`Text`** (Course Dir, const Real *A, const char* _fmt, ...) *// координатах*

Текст печатается на графическом поле, при этом выбор **Course** показывает отступ текста от контрольной точки X,Y,Z на поверхности рабочей площадки:

Center – указанные координаты приходятся на центр надписи;
North – со смещением вверх; **South** – вниз;
West – влево, **East** – вправо.

Для справки приводятся числовые коды некоторых символов, которые нередко срабатываю на клавиатуре при нажатии клавиши <flt>.

1. ☺	11 ♂	21 §	31 ▼	0157	0169 ©	0179 ³	0189 ½
2. ☻	12 ♀	22 ▬	0145 ‘	0160	0170 ª	0180 ´	0190 ¾
3. ♥	13 ♪	23 ↕	0146 ’	0161 ¡	0171 «	0181 µ	0191 ¿
4. ♦	14 ♫	24 ↑	0147 “	0162 ¢	0172 ¬	0182 ¶	0215 ×
5. ♣	15 ☼	25 ↓	0148 ”	0163 £	0173	0183 ·	0216 Ø
6. ♠	16 ►	26 →	0149 •	0164 ¤	0174 ®	0184 ¸	0223 ß
7. •	17 ◄	27 ←	0151 —	0165 ¥	0175 ¯	0185 ¹	0230 æ
8. ◘	18 ↕	28 ∟	0152 ˜	0166 ¦	0176 °	0186 º	0247 ÷
9. ○	19 ‼	29 ↔	0153 ™	0167 §	0177 ±	0187 »	0248 ø
10 ◙	20 ¶	30 ▲	0155 ›	0168 ¨	0178 ²	0188 ¼	

В стандартной русской кодировке Windows-1251 имеются специальные символы, которые со временем планируются к включению в расширенную матрицу DispCCCP:

`'°' – B0` **`'Ё' – A8`** `'©'` – A9 `'§'` – A7
`'±' – B1` **`'ё' – B8`** `'®'` – AE **`'«' – AB`**
`'≠'` – 87 `'№'` – B9 `'™'` – 99 **`'»' – BB`** ... 86

Последняя кодовая строка из альтернативной (OEM-866) кодировки в DOS, перенесена в позицию 0x80 и содержит следующие символы: **`'`**`≡±≥≤⌠⌡÷≈°•·√ⁿ²■`**`'`**`,` устанавливаемые функцией для старого русского растра **`AlfaBit`**`( _8x08 | _8x14 | _8x16 )`<=Win1251.

Две процедуры для наложения текстовых меню и подсказок

```
 //! Help – текстовое окошко для кратких подсказок с управляющими кодами программы
 //  N[0]    – подзаголовок – краткое обозначение набора инструкций для Window
 //  N[1-3]  – три строки расширенного названия окна подсказок (~~и признак движения~~)
 //  Cmd     – парное описание команд или операций с кратким предназначением
 //  Plus    – то же для блока дополнительных инструкций и подсказок
 //  ++ определение каждого блока текстовых строк заканчивается нулевым адресом

void Window::Help
  ( const char *Heading[], // [0,1÷3] заголовок и строки расширенного названия
    const char *Commands[], //  парное описание основных команд и действий
    const char *Comments[], // + всякие дополнительные парные примечания
    int X=-1, int Y=1        // = местоположение в окне { -1–справа 1–вверху }
  );

struct Mlist{ short skip,lf; const char *Msg; void *dat; };
#define Mlist( L ) L,( sizeof( L )/sizeof( Mlist ) ) // строка и ее длина

//      Mlist – список параметров для одного запроса текстового меню на терминал
//         skip  :  пропуск строк                —> номер сроки
//         lf    :  0 – запрос не производится   —> длина входного сообщения
//         Msg   :  NULL – чистое входное поле   —> выходной формат –
//         dat   :  NULL & lf<>0 – меню-запрос   —> адрес изменяемого объекта

int TMenu( Mlist *M, int Nm, int x=1, int y=1, int ans=0 );

class TextMenu: Place // запрос текстового меню с отсрочкой полного завершения
{ int Y,X,Lx,Ly,   // местоположение на экране (++/слева-сверху, --/снизу-справа)
      K,           // номер редактируемого поля / последнего обращения
      Num;         // количество строк меню
  Mlist *M;        // собственно список меню Mlist/mlist
// void(*)(int);   // прерывание/подсказка при переходе на новый запрос из меню
  bool Up;         // признак установки меню на экране
public:
  TextMenu( Mlist*,int, int=1,int=1 ); ~TextMenu();
  void Active(); // локальная активизация графического контекста новой площадки
  int Answer( int=-1 ); void Back(){ Up=false; Free(); }
};
```

Операции C++

По порядку слева → направо, или налево ← справа,
с приоритетом от 17 до 1 по старшинству операций.

Первичные и постфиксные

`::` →17 *разрешение области видимости*
`[]` →16 *индексация массива*
`()` →16 *вызов функции*
`.` →16 *элемент структуры*
`->` →16 *элемент указателя*
`++` →15 *постфиксный инкремент*
`--` →15 *постфиксный декремент*

Одноместные операции

`++` ←14 *префиксный инкремент*
`--` ←14 *префиксный декремент*
`~` ←14 *поразрядное NOT*
`!` ←14 *логическое NOT*
`-` ←14 *унарный минус*
`&` ←14 *взятие адреса*
`*` ←14 *разыменование указателя*
`::*` ←14 *указатель на член класса*
`.*` ←14 *доступ к члену класса*
`->*` ←14 *доступ по ссылке —//—*
(*тип*) ←14 *приведение типа*
`sizeof` ←14 *размер в байтах*

Мультипликативные

`*` →13 *умножение*
`/` →13 *деление*
`%` →13 *взятие по модулю*

Аддитивные

`+` →12 *сложение*
`-` →12 *вычитание*

Поразрядного сдвига

`<<` →11 *сдвиг влево*
`>>` →11 *сдвиг вправо*

Отношения

`<` →10 *меньше*
`<=` →10 *меньше или равно*
`>` →10 *больше*
`>=` →10 *больше или равно*
`==` → 9 *равно*
`!=` → 9 *не равно*

Поразрядные

`&` → 8 *поразрядное* AND
`^` → 7 *поразрядное* XOR
`|` → 6 *поразрядное* OR

Логические

`&&` → 5 *логическое* AND
`||` → 4 *логическое* OR

Условные

`? :` ← 3 *условная операция*

Присваивания

`=` ← 2 *присваивание*
`*=` ← 2 *присвоение произведения*
`/=` ← 2 *присвоение частного*
`%=` ← 2 *присвоение модуля*
`+=` ← 2 *присвоение суммы*
`-=` ← 2 *присвоение разности*
`<<=` ← 2 *присвоение левого сдвига*
`>>=` ← 2 *присвоение правого сдвига*
`&=` ← 2 *присвоение* AND
`^=` ← 2 *присвоение* XOR
`|=` ← 2 *присвоение* OR
`throw` ← 2 *исключение*
`,` → 1 *запятая*

Вычислительная гидромеханика и морские исследования
Сахалинский государственный университет