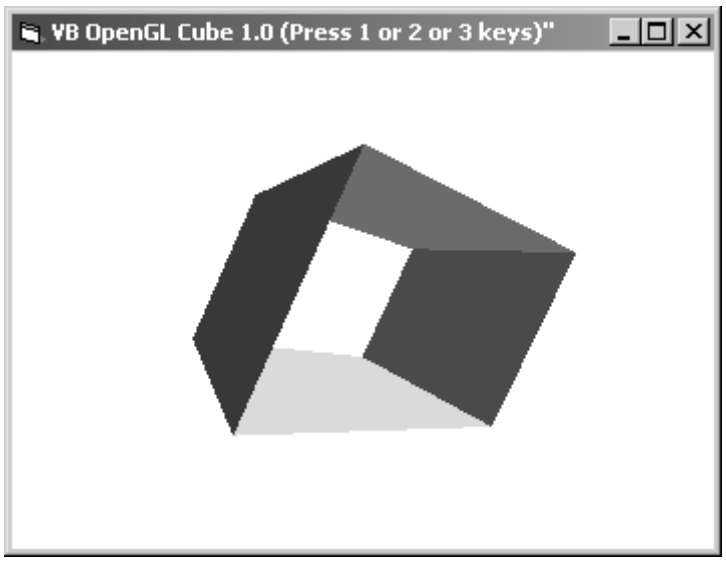
فصل بیست و نهم :

بکارگیری OpenGL در ویژوال بیسیک ۶



مقدمه :

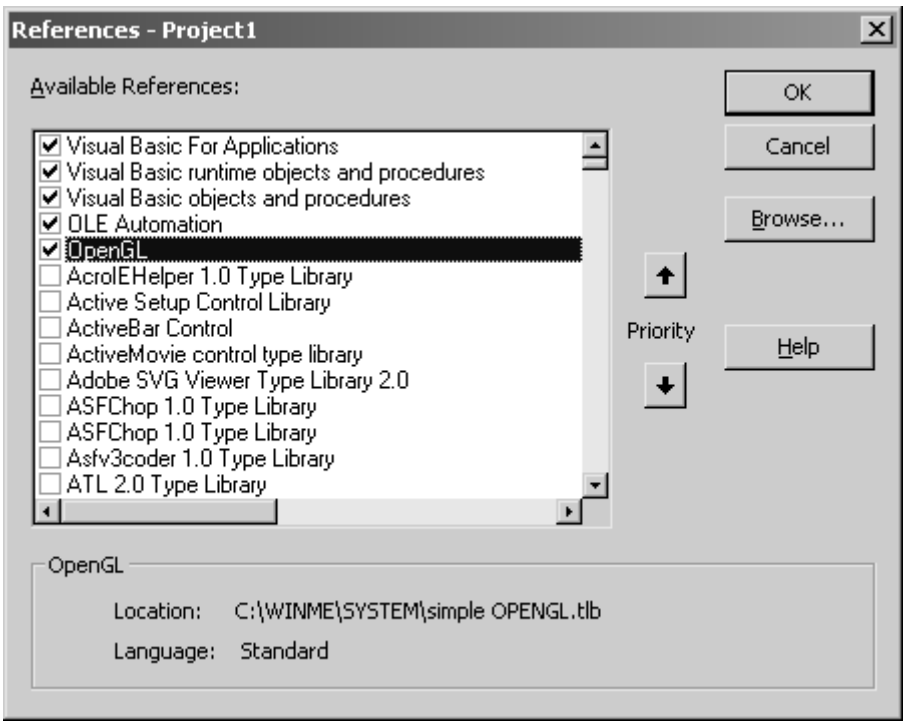
همانند واحدهای دلفی و Header Files زبان C/C++ ، ویژوال بیسیک نیز با استفاده از Type Libraries می تواند به محتویات فایل های OCX و یا DLL دسترسی مطمئنی پیدا کند. بعضی از فایل های DLL خود دارای Type Library می باشند مانند Active-X Dll's ، ولی فایل های Dll ایی که Type Library بصورت Resource در درون خودشان ویا مجزا ، ندارند؛ بسادگی قابل استفاده در VB نمی باشند . بدین جهت برای آنها باید فایل Type Library لازم (*.tlb) را ایجاد نمود .
خود VB ابزاری را برای این کار ندارد ، ولی با استفاده از دلفی و یا VC++ اینکار را می توان انجام داد که در ادامه به شرح آن پرداخته می شود . قبل از شروع به توضیح در مورد فایل های tlb لازم به ذکر است که اگر به آدرس زیر مراجعه فرمایید ، فایل tlb مخصوص OpenGL برای استفاده در VB یا هر کامپایلر دیگری را می توانید دریافت کنید و از خواندن این فصل در آن صورت می توان صرفنظر کرد !

http://www.chez.com/scribe/

Type Library **چیست** ؟

Tlb یک متن کامپایل و نوشته شده به زبان تعریف شیءایی (Object Definition Language (ODL)) می باشد که حاوی توضیحاتی درمورد نوع های داده ای و اشیاء موجود در فایل های Dll ، OCX یا Exe بوده و برای کار در اتوماسیون OLE بکار گرفته می شوند . کد لازم برای ایجاد tlb از ترجمه Header files ، فایل DLL که معمولا به همراه آن ارئه می گردد ، ایجاد می شود . سپس این کد توسط کامپایلر MIDL ، کامپایل می شود .

در ادامه فایل tlb حاصل باید در ویندوز ثبت شود . برای این کار می توان به منوی Project-References... > مراجعه کرد (شکل–۱). پس از ثبت tlb هر برنامه ای که در VB بخواهد از آن DLL استفاده کند فقط کافی است تا در منوی یاد شده ، این Tlb را انتخاب کند.پس از انتخاب tlb لازم ، تمام اطلاعات موجود در آن، در محیط VB به راحتی قابل استفاده است و دیگری نیازی به تعریف روتین های Dll بصورت دستی در ماژول های VB نخواهد بود . برای مثال تمام Win32 API در سایت فوق بصورت یک tlb در آمده است و دیگری نیازی به تعریف دستی روتین های API ویندوز در VB نخواهد بود .



شکل–۱ – طریقه ثبت و استفاده از Tlb ایجاد شده در VB .

طریقه ایجاد Type Libraries :

در زیر طریقه ایجاد یک فایل ODL از یک Header File (*.h) زبان C و کامپایل آن به یک فایل Tlb توضیح داده خواهد شد :

۱- مرحله اول ترجمه کردن h.* به odl.* می باشد . برای این کار باید تمام typedef ها به ابتدای فایل اضافه شوند ، سپس تمام #define ها به const تبدیل و [... entery] به ابتدای تمام تعاریف توابع اضافه می شوند . سپس باید برای این فایل یک شماره منحصر بفرد : GUID ، ایجاد نمود و در نهایت فایل حاصل چیزی شبیه به متن زیر خواهد بود :

```
//library attributes:
[ uuid(03644881-8010-11d1-95AA-000000000000),
helpstring("VB OpenGL API 1.1 (ANSI)")
]
library VBOpenGL
{
//typedefs ...
//module attributes:
[
uuid(1D15DCA0-802D-11d1-95AA-000000000000),
helpstring("OpenGL gl functions"),
dllname("OPENGL32")
]
module GL
{
//constants:
const int GL_BYTE = 0x1400;
//function prototypes:
[entry("glColor3f"),
helpstring("Sets the current color"),]
void APIENTRY glColor3f (GLfloat red,GLfloat green, GLfloat blue);
}
//more typedefs ...
//more modules ...
}
```

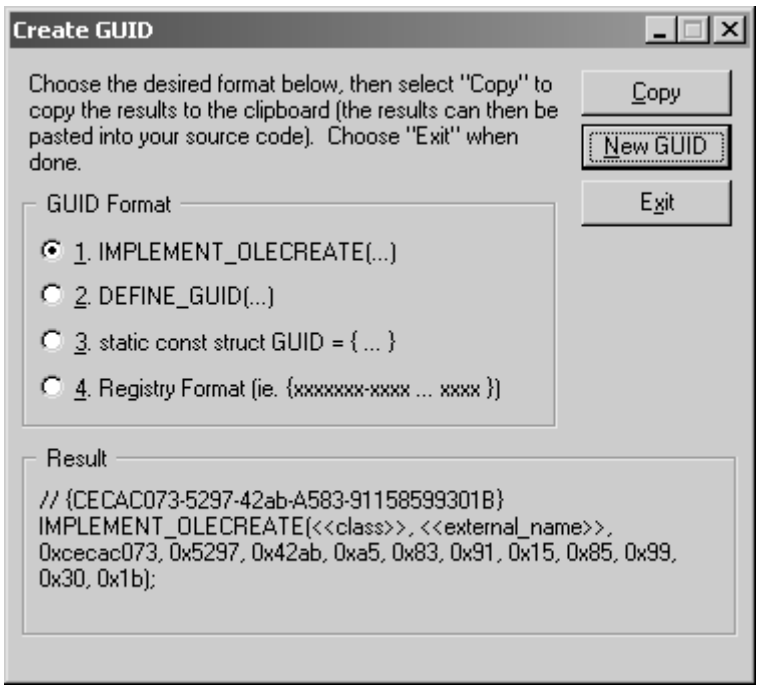
۲- اصلاح بعضی از تعاریف تا در VB و خصوصا OLE قابل قبول باشند . بعضی از نوع های داده ای زبان C در VB قابل قبول نمی باشند ، برای مثال :

VB اشاره گر به توابع و یا اشاره گر به اشاره گرها را نمی پذیرد . این مشکل را با تغییر پارامتر به Long و یا LPVOID می توان حل کرد .

رشته ها : بعضی از توابع API حاوی رشته هایی با طول متغیر هستند که در tlb قابل تعریف نمی باشند و حتما باید بصورت جداگانه در VB تعریف شوند .

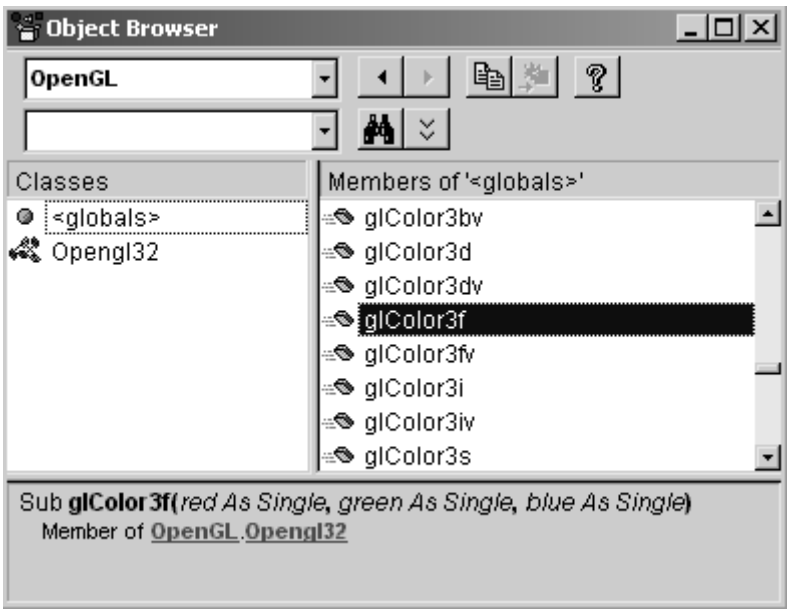
همواره فراخوانی روتین های C از درون VB مشکل زا بوده است و یکی از اهداف تکنولوژی COM رفع اینگونه مشکلات می باشد .

Header files مربوط به OpenGL را می توانید از سی دی ویژوال استودیو مایکروسافت بدست آورید و همچنین برنامه های Mktyplib.exe ، Oleview.exe و Guidgen.exe که به ترتیب برای **کامپایل** کردن فایل odl به tlb ، مشاهده درون فایل های tlb و تولید عدد منحصر بفرد شناسایی فایل tlb بکار می روند . البته دلفی نیز ابزارهای مشابهی را ارائه داده است ، برای مثال : برای ایجاد فایل tlb فقط کافی است به مسیر File Menu->New->Active-X->tlb ... مراجعه کنید و یا اگر مایل به مشاهده درون فایلهای کامپایل شده tlb هستید فقط کافی است به مسیر File Menu->Open مراجعه نمایید و ... .



شکل ۲– Guidgen.exe که برای تولید عدد منحصر بفرد شناسایی tlb بکار می رود.

```
Private Sub Form_Load()
glcolor3f(
```
glColor3f(***red As Single***, *green As Single*, *blue As Single*)



شکل ۳- جستجوگر شیء در VB . پس از ثبت و انتخاب tlb لازم می توان از آن استفاده کرد.

در زیر کد کامل یک tlb نمونه برای بکار گیری OpenGL در VB رائه می گردد :

```
[
  uuid(27B854C0-C09B-11D0-8253-00805F2A72AB)
]
library OpenGL
{

  importlib("stdole32.tlb");
  importlib("olepro32.dll");
  importlib("STDVCL32.DLL");

  [
    uuid(27B854C1-C09B-11D0-8253-00805F2A72AB),
    dllname("opengl32.dll")
  ]
   module Opengl32
  {
    const long GL_VERSION_1_1 = 1;
    const long GL_ACCUM = 256;
    const long GL_LOAD = 257;
    const long GL_RETURN = 258;
    const long GL_MULT = 259;
    const long GL_ADD = 260;
    const long GL_NEVER = 512;
    const long GL_LESS = 513;
    const long GL_EQUAL = 514;
    const long GL_LEQUAL = 515;
    const long GL_GREATER = 516;
    const long GL_NOTEQUAL = 517;
    const long GL_GEQUAL = 518;
    const long GL_ALWAYS = 519;
    const long GL_CURRENT_BIT = 1;
```

```
const long GL_POINT_BIT = 2;
const long GL_LINE_BIT = 4;
const long GL_POLYGON_BIT = 8;
const long GL_POLYGON_STIPPLE_BIT = 16;
const long GL_PIXEL_MODE_BIT = 32;
const long GL_LIGHTING_BIT = 64;
const long GL_FOG_BIT = 128;
const long GL_DEPTH_BUFFER_BIT = 256;
const long GL_ACCUM_BUFFER_BIT = 512;
const long GL_STENCIL_BUFFER_BIT = 1024;
const long GL_VIEWPORT_BIT = 2048;
const long GL_TRANSFORM_BIT = 4096;
const long GL_ENABLE_BIT = 8192;
const long GL_COLOR_BUFFER_BIT = 16384;
const long GL_HINT_BIT = 32768;
const long GL_EVAL_BIT = 65536;
const long GL_LIST_BIT = 131072;
const long GL_TEXTURE_BIT = 262144;
const long GL_SCISSOR_BIT = 524288;
const long GL_ALL_ATTRIB_BITS = 1048575;
const long GL_POINTS = 0;
const long GL_LINES = 1;
const long GL_LINE_LOOP = 2;
const long GL_LINE_STRIP = 3;
const long GL_TRIANGLES = 4;
const long GL_TRIANGLE_STRIP = 5;
const long GL_TRIANGLE_FAN = 6;
const long GL_QUADS = 7;
const long GL_QUAD_STRIP = 8;
const long GL_POLYGON = 9;
const long GL_ZERO = 0;
const long GL_ONE = 1;
const long GL_SRC_COLOR = 768;
const long GL_ONE_MINUS_SRC_COLOR = 769;
const long GL_SRC_ALPHA = 770;
const long GL_ONE_MINUS_SRC_ALPHA = 771;
const long GL_DST_ALPHA = 772;
const long GL_ONE_MINUS_DST_ALPHA = 773;
const long GL_DST_COLOR = 774;
const long GL_ONE_MINUS_DST_COLOR = 775;
const long GL_SRC_ALPHA_SATURATE = 776;
const long GL_TRUE = 1;
const long GL_FALSE = 0;
const long GL_CLIP_PLANE0 = 12288;
const long GL_CLIP_PLANE1 = 12289;
const long GL_CLIP_PLANE2 = 12290;
const long GL_CLIP_PLANE3 = 12291;
const long GL_CLIP_PLANE4 = 12292;
const long GL_CLIP_PLANE5 = 12293;
const long GL_BYTE = 5120;
const long GL_UNSIGNED_BYTE = 5121;
const long GL_SHORT = 5122;
const long GL_UNSIGNED_SHORT = 5123;
const long GL_INT = 5124;
const long GL_UNSIGNED_INT = 5125;
const long GL_FLOAT = 5126;
const long GL_2_BYTES = 5127;
const long GL_3_BYTES = 5128;
```

```
const long GL_4_BYTES = 5129;
const long GL_DOUBLE = 5130;
const long GL_NONE = 0;
const long GL_FRONT_LEFT = 1024;
const long GL_FRONT_RIGHT = 1025;
const long GL_BACK_LEFT = 1026;
const long GL_BACK_RIGHT = 1027;
const long GL_FRONT = 1028;
const long GL_BACK = 1029;
const long GL_LEFT = 1030;
const long GL_RIGHT = 1031;
const long GL_FRONT_AND_BACK = 1032;
const long GL_AUX0 = 1033;
const long GL_AUX1 = 1034;
const long GL_AUX2 = 1035;
const long GL_AUX3 = 1036;
const long GL_NO_ERROR = 0;
const long GL_INVALID_ENUM = 1280;
const long GL_INVALID_VALUE = 1281;
const long GL_INVALID_OPERATION = 1282;
const long GL_STACK_OVERFLOW = 1283;
const long GL_STACK_UNDERFLOW = 1284;
const long GL_OUT_OF_MEMORY = 1285;
const long GL_2D = 1536;
const long GL_3D = 1537;
const long GL_3D_COLOR = 1538;
const long GL_3D_COLOR_TEXTURE = 1539;
const long GL_4D_COLOR_TEXTURE = 1540;
const long GL_PASS_THROUGH_TOKEN = 1792;
const long GL_POINT_TOKEN = 1793;
const long GL_LINE_TOKEN = 1794;

const long GL_POLYGON_TOKEN = 1795;
const long GL_BITMAP_TOKEN = 1796;
const long GL_DRAW_PIXEL_TOKEN = 1797;
const long GL_COPY_PIXEL_TOKEN = 1798;
const long GL_LINE_RESET_TOKEN = 1799;
const long GL_EXP = 2048;
const long GL_EXP2 = 2049;
const long GL_CW = 2304;
const long GL_CCW = 2305;
const long GL_COEFF = 2560;
const long GL_ORDER = 2561;
const long GL_DOMAIN = 2562;
const long GL_CURRENT_COLOR = 2816;
const long GL_CURRENT_INDEX = 2817;
const long GL_CURRENT_NORMAL = 2818;
const long GL_CURRENT_TEXTURE_COORDS = 2819;
const long GL_CURRENT_RASTER_COLOR = 2820;
const long GL_CURRENT_RASTER_INDEX = 2821;
const long GL_CURRENT_RASTER_TEXTURE_COORDS = 2822;
const long GL_CURRENT_RASTER_POSITION = 2823;
const long GL_CURRENT_RASTER_POSITION_VALID = 2824;
const long GL_CURRENT_RASTER_DISTANCE = 2825;
const long GL_POINT_SMOOTH = 2832;
const long GL_POINT_SIZE = 2833;
const long GL_POINT_SIZE_RANGE = 2834;
const long GL_POINT_SIZE_GRANULARITY = 2835;
```

```
const long GL_LINE_SMOOTH = 2848;
const long GL_LINE_WIDTH = 2849;
const long GL_LINE_WIDTH_RANGE = 2850;
const long GL_LINE_WIDTH_GRANULARITY = 2851;
const long GL_LINE_STIPPLE = 2852;
const long GL_LINE_STIPPLE_PATTERN = 2853;
const long GL_LINE_STIPPLE_REPEAT = 2854;
const long GL_LIST_MODE = 2864;
const long GL_MAX_LIST_NESTING = 2865;
const long GL_LIST_BASE = 2866;
const long GL_LIST_INDEX = 2867;
const long GL_POLYGON_MODE = 2880;
const long GL_POLYGON_SMOOTH = 2881;
const long GL_POLYGON_STIPPLE = 2882;
const long GL_EDGE_FLAG = 2883;
const long GL_CULL_FACE = 2884;
const long GL_CULL_FACE_MODE = 2885;
const long GL_FRONT_FACE = 2886;
const long GL_LIGHTING = 2896;
const long GL_LIGHT_MODEL_LOCAL_VIEWER = 2897;
const long GL_LIGHT_MODEL_TWO_SIDE = 2898;
const long GL_LIGHT_MODEL_AMBIENT = 2899;
const long GL_SHADE_MODEL = 2900;
const long GL_COLOR_MATERIAL_FACE = 2901;
const long GL_COLOR_MATERIAL_PARAMETER = 2902;
const long GL_COLOR_MATERIAL = 2903;
const long GL_FOG = 2912;
const long GL_FOG_INDEX = 2913;
const long GL_FOG_DENSITY = 2914;
const long GL_FOG_START = 2915;
const long GL_FOG_END = 2916;
const long GL_FOG_MODE = 2917;
const long GL_FOG_COLOR = 2918;
const long GL_DEPTH_RANGE = 2928;
const long GL_DEPTH_TEST = 2929;
const long GL_DEPTH_WRITEMASK = 2930;
const long GL_DEPTH_CLEAR_VALUE = 2931;
const long GL_DEPTH_FUNC = 2932;
const long GL_ACCUM_CLEAR_VALUE = 2944;
const long GL_STENCIL_TEST = 2960;
const long GL_STENCIL_CLEAR_VALUE = 2961;
const long GL_STENCIL_FUNC = 2962;
const long GL_STENCIL_VALUE_MASK = 2963;
const long GL_STENCIL_FAIL = 2964;
const long GL_STENCIL_PASS_DEPTH_FAIL = 2965;
const long GL_STENCIL_PASS_DEPTH_PASS = 2966;
const long GL_STENCIL_REF = 2967;
const long GL_STENCIL_WRITEMASK = 2968;
const long GL_MATRIX_MODE = 2976;
const long GL_NORMALIZE = 2977;
const long GL_VIEWPORT = 2978;
const long GL_MODELVIEW_STACK_DEPTH = 2979;
const long GL_PROJECTION_STACK_DEPTH = 2980;
const long GL_TEXTURE_STACK_DEPTH = 2981;
const long GL_MODELVIEW_MATRIX = 2982;
const long GL_PROJECTION_MATRIX = 2983;
const long GL_TEXTURE_MATRIX = 2984;
const long GL_ATTRIB_STACK_DEPTH = 2992;
```

```
const long GL_CLIENT_ATTRIB_STACK_DEPTH = 2993;
const long GL_ALPHA_TEST = 3008;
const long GL_ALPHA_TEST_FUNC = 3009;
const long GL_ALPHA_TEST_REF = 3010;
const long GL_DITHER = 3024;
const long GL_BLEND_DST = 3040;
const long GL_BLEND_SRC = 3041;
const long GL_BLEND = 3042;
const long GL_LOGIC_OP_MODE = 3056;
const long GL_INDEX_LOGIC_OP = 3057;
const long GL_COLOR_LOGIC_OP = 3058;
const long GL_AUX_BUFFERS = 3072;
const long GL_DRAW_BUFFER = 3073;
const long GL_READ_BUFFER = 3074;
const long GL_SCISSOR_BOX = 3088;
const long GL_SCISSOR_TEST = 3089;
const long GL_INDEX_CLEAR_VALUE = 3104;
const long GL_INDEX_WRITEMASK = 3105;
const long GL_COLOR_CLEAR_VALUE = 3106;
const long GL_COLOR_WRITEMASK = 3107;
const long GL_INDEX_MODE = 3120;
const long GL_RGBA_MODE = 3121;
const long GL_DOUBLEBUFFER = 3122;
const long GL_STEREO = 3123;
const long GL_RENDER_MODE = 3136;
const long GL_PERSPECTIVE_CORRECTION_HINT = 3152;
const long GL_POINT_SMOOTH_HINT = 3153;
const long GL_LINE_SMOOTH_HINT = 3154;
const long GL_POLYGON_SMOOTH_HINT = 3155;
const long GL_FOG_HINT = 3156;
const long GL_TEXTURE_GEN_S = 3168;
const long GL_TEXTURE_GEN_T = 3169;
const long GL_TEXTURE_GEN_R = 3170;
const long GL_TEXTURE_GEN_Q = 3171;
const long GL_PIXEL_MAP_I_TO_I = 3184;
const long GL_PIXEL_MAP_S_TO_S = 3185;
const long GL_PIXEL_MAP_I_TO_R = 3186;
const long GL_PIXEL_MAP_I_TO_G = 3187;
const long GL_PIXEL_MAP_I_TO_B = 3188;
const long GL_PIXEL_MAP_I_TO_A = 3189;
const long GL_PIXEL_MAP_R_TO_R = 3190;
const long GL_PIXEL_MAP_G_TO_G = 3191;
const long GL_PIXEL_MAP_B_TO_B = 3192;
const long GL_PIXEL_MAP_A_TO_A = 3193;
const long GL_PIXEL_MAP_I_TO_I_SIZE = 3248;
const long GL_PIXEL_MAP_S_TO_S_SIZE = 3249;
const long GL_PIXEL_MAP_I_TO_R_SIZE = 3250;
const long GL_PIXEL_MAP_I_TO_G_SIZE = 3251;
const long GL_PIXEL_MAP_I_TO_B_SIZE = 3252;
const long GL_PIXEL_MAP_I_TO_A_SIZE = 3253;
const long GL_PIXEL_MAP_R_TO_R_SIZE = 3254;
const long GL_PIXEL_MAP_G_TO_G_SIZE = 3255;
const long GL_PIXEL_MAP_B_TO_B_SIZE = 3256;
const long GL_PIXEL_MAP_A_TO_A_SIZE = 3257;
const long GL_UNPACK_SWAP_BYTES = 3312;
const long GL_UNPACK_LSB_FIRST = 3313;
const long GL_UNPACK_ROW_LENGTH = 3314;
const long GL_UNPACK_SKIP_ROWS = 3315;
```

```
const long GL_UNPACK_SKIP_PIXELS = 3316;
const long GL_UNPACK_ALIGNMENT = 3317;
const long GL_PACK_SWAP_BYTES = 3328;
const long GL_PACK_LSB_FIRST = 3329;
const long GL_PACK_ROW_LENGTH = 3330;
const long GL_PACK_SKIP_ROWS = 3331;
const long GL_PACK_SKIP_PIXELS = 3332;
const long GL_PACK_ALIGNMENT = 3333;
const long GL_MAP_COLOR = 3344;
const long GL_MAP_STENCIL = 3345;
const long GL_INDEX_SHIFT = 3346;
const long GL_INDEX_OFFSET = 3347;
const long GL_RED_SCALE = 3348;
const long GL_RED_BIAS = 3349;
const long GL_ZOOM_X = 3350;
const long GL_ZOOM_Y = 3351;
const long GL_GREEN_SCALE = 3352;
const long GL_GREEN_BIAS = 3353;
const long GL_BLUE_SCALE = 3354;
const long GL_BLUE_BIAS = 3355;
const long GL_ALPHA_SCALE = 3356;
const long GL_ALPHA_BIAS = 3357;
const long GL_DEPTH_SCALE = 3358;
const long GL_DEPTH_BIAS = 3359;
const long GL_MAX_EVAL_ORDER = 3376;
const long GL_MAX_LIGHTS = 3377;
const long GL_MAX_CLIP_PLANES = 3378;
const long GL_MAX_TEXTURE_SIZE = 3379;
const long GL_MAX_PIXEL_MAP_TABLE = 3380;
const long GL_MAX_ATTRIB_STACK_DEPTH = 3381;
const long GL_MAX_MODELVIEW_STACK_DEPTH = 3382;
const long GL_MAX_NAME_STACK_DEPTH = 3383;
const long GL_MAX_PROJECTION_STACK_DEPTH = 3384;
const long GL_MAX_TEXTURE_STACK_DEPTH = 3385;
const long GL_MAX_VIEWPORT_DIMS = 3386;
const long GL_MAX_CLIENT_ATTRIB_STACK_DEPTH = 3387;
const long GL_SUBPIXEL_BITS = 3408;
const long GL_INDEX_BITS = 3409;
const long GL_RED_BITS = 3410;
const long GL_GREEN_BITS = 3411;
const long GL_BLUE_BITS = 3412;
const long GL_ALPHA_BITS = 3413;
const long GL_DEPTH_BITS = 3414;
const long GL_STENCIL_BITS = 3415;
const long GL_ACCUM_RED_BITS = 3416;
const long GL_ACCUM_GREEN_BITS = 3417;
const long GL_ACCUM_BLUE_BITS = 3418;
const long GL_ACCUM_ALPHA_BITS = 3419;
const long GL_NAME_STACK_DEPTH = 3440;
const long GL_AUTO_NORMAL = 3456;
const long GL_MAP1_COLOR_4 = 3472;
const long GL_MAP1_INDEX = 3473;
const long GL_MAP1_NORMAL = 3474;
const long GL_MAP1_TEXTURE_COORD_1 = 3475;
const long GL_MAP1_TEXTURE_COORD_2 = 3476;
const long GL_MAP1_TEXTURE_COORD_3 = 3477;
const long GL_MAP1_TEXTURE_COORD_4 = 3478;
const long GL_MAP1_VERTEX_3 = 3479;
```

```
const long GL_MAP1_VERTEX_4 = 3480;
const long GL_MAP2_COLOR_4 = 3504;
const long GL_MAP2_INDEX = 3505;
const long GL_MAP2_NORMAL = 3506;
const long GL_MAP2_TEXTURE_COORD_1 = 3507;
const long GL_MAP2_TEXTURE_COORD_2 = 3508;
const long GL_MAP2_TEXTURE_COORD_3 = 3509;
const long GL_MAP2_TEXTURE_COORD_4 = 3510;
const long GL_MAP2_VERTEX_3 = 3511;
const long GL_MAP2_VERTEX_4 = 3512;
const long GL_MAP1_GRID_DOMAIN = 3536;
const long GL_MAP1_GRID_SEGMENTS = 3537;
const long GL_MAP2_GRID_DOMAIN = 3538;
const long GL_MAP2_GRID_SEGMENTS = 3539;
const long GL_TEXTURE_1D = 3552;
const long GL_TEXTURE_2D = 3553;
const long GL_FEEDBACK_BUFFER_POINTER = 3568;
const long GL_FEEDBACK_BUFFER_SIZE = 3569;
const long GL_FEEDBACK_BUFFER_TYPE = 3570;
const long GL_SELECTION_BUFFER_POINTER = 3571;
const long GL_SELECTION_BUFFER_SIZE = 3572;
const long GL_TEXTURE_WIDTH = 4096;
const long GL_TEXTURE_HEIGHT = 4097;
const long GL_TEXTURE_INTERNAL_FORMAT = 4099;
const long GL_TEXTURE_BORDER_COLOR = 4100;
const long GL_TEXTURE_BORDER = 4101;
const long GL_DONT_CARE = 4352;
const long GL_FASTEST = 4353;
const long GL_NICEST = 4354;
const long GL_LIGHT0 = 16384;
const long GL_LIGHT1 = 16385;
const long GL_LIGHT2 = 16386;
const long GL_LIGHT3 = 16387;
const long GL_LIGHT4 = 16388;
const long GL_LIGHT5 = 16389;
const long GL_LIGHT6 = 16390;
const long GL_LIGHT7 = 16391;
const long GL_AMBIENT = 4608;
const long GL_DIFFUSE = 4609;
const long GL_SPECULAR = 4610;
const long GL_POSITION = 4611;
const long GL_SPOT_DIRECTION = 4612;
const long GL_SPOT_EXPONENT = 4613;
const long GL_SPOT_CUTOFF = 4614;
const long GL_CONSTANT_ATTENUATION = 4615;
const long GL_LINEAR_ATTENUATION = 4616;
const long GL_QUADRATIC_ATTENUATION = 4617;
const long GL_COMPILE = 4864;
const long GL_COMPILE_AND_EXECUTE = 4865;
const long GL_CLEAR = 5376;
const long GL_AND = 5377;
const long GL_AND_REVERSE = 5378;
const long GL_COPY = 5379;
const long GL_AND_INVERTED = 5380;
const long GL_NOOP = 5381;
const long GL_XOR = 5382;
const long GL_OR = 5383;
const long GL_NOR = 5384;
```

```
const long GL_EQUIV = 5385;
const long GL_INVERT = 5386;
const long GL_OR_REVERSE = 5387;
const long GL_COPY_INVERTED = 5388;
const long GL_OR_INVERTED = 5389;
const long GL_NAND = 5390;
const long GL_SET = 5391;
const long GL_EMISSION = 5632;
const long GL_SHININESS = 5633;
const long GL_AMBIENT_AND_DIFFUSE = 5634;
const long GL_COLOR_INDEXES = 5635;
const long GL_MODELVIEW = 5888;
const long GL_PROJECTION = 5889;
const long GL_TEXTURE = 5890;
const long GL_COLOR = 6144;
const long GL_DEPTH = 6145;
const long GL_STENCIL = 6146;
const long GL_COLOR_INDEX = 6400;
const long GL_STENCIL_INDEX = 6401;
const long GL_DEPTH_COMPONENT = 6402;
const long GL_RED = 6403;
const long GL_GREEN = 6404;
const long GL_BLUE = 6405;
const long GL_ALPHA = 6406;
const long GL_RGB = 6407;
const long GL_RGBA = 6408;
const long GL_LUMINANCE = 6409;
const long GL_LUMINANCE_ALPHA = 6410;
const long GL_BITMAP = 6656;
const long GL_POINT = 6912;
const long GL_LINE = 6913;
const long GL_FILL = 6914;
const long GL_RENDER = 7168;
const long GL_FEEDBACK = 7169;
const long GL_SELECT = 7170;
const long GL_FLAT = 7424;
const long GL_SMOOTH = 7425;
const long GL_KEEP = 7680;
const long GL_REPLACE = 7681;
const long GL_INCR = 7682;
const long GL_DECR = 7683;
const long GL_VENDOR = 7936;
const long GL_RENDERER = 7937;
const long GL_VERSION = 7938;
const long GL_EXTENSIONS = 7939;
const long GL_S = 8192;
const long GL_T = 8193;
const long GL_R = 8194;
const long GL_Q = 8195;
const long GL_MODULATE = 8448;
const long GL_DECAL = 8449;
const long GL_TEXTURE_ENV_MODE = 8704;
const long GL_TEXTURE_ENV_COLOR = 8705;
const long GL_TEXTURE_ENV = 8960;
const long GL_EYE_LINEAR = 9216;
const long GL_OBJECT_LINEAR = 9217;
const long GL_SPHERE_MAP = 9218;
const long GL_TEXTURE_GEN_MODE = 9472;
```

```
const long GL_OBJECT_PLANE = 9473;
const long GL_EYE_PLANE = 9474;
const long GL_NEAREST = 9728;
const long GL_LINEAR = 9729;
const long GL_NEAREST_MIPMAP_NEAREST = 9984;
const long GL_LINEAR_MIPMAP_NEAREST = 9985;
const long GL_NEAREST_MIPMAP_LINEAR = 9986;
const long GL_LINEAR_MIPMAP_LINEAR = 9987;
const long GL_TEXTURE_MAG_FILTER = 10240;
const long GL_TEXTURE_MIN_FILTER = 10241;
const long GL_TEXTURE_WRAP_S = 10242;
const long GL_TEXTURE_WRAP_T = 10243;
const long GL_CLAMP = 10496;
const long GL_REPEAT = 10497;
const long GL_CLIENT_PIXEL_STORE_BIT = 1;
const long GL_CLIENT_VERTEX_ARRAY_BIT = 2;
const long GL_CLIENT_ALL_ATTRIB_BITS = -1;
const long GL_POLYGON_OFFSET_FACTOR = 32824;
const long GL_POLYGON_OFFSET_UNITS = 10752;
const long GL_POLYGON_OFFSET_POINT = 10753;
const long GL_POLYGON_OFFSET_LINE = 10754;
const long GL_POLYGON_OFFSET_FILL = 32823;
const long GL_ALPHA4 = 32827;
const long GL_ALPHA8 = 32828;
const long GL_ALPHA12 = 32829;
const long GL_ALPHA16 = 32830;
const long GL_LUMINANCE4 = 32831;
const long GL_LUMINANCE8 = 32832;
const long GL_LUMINANCE12 = 32833;
const long GL_LUMINANCE16 = 32834;
const long GL_LUMINANCE4_ALPHA4 = 32835;
const long GL_LUMINANCE6_ALPHA2 = 32836;
const long GL_LUMINANCE8_ALPHA8 = 32837;
const long GL_LUMINANCE12_ALPHA4 = 32838;
const long GL_LUMINANCE12_ALPHA12 = 32839;
const long GL_LUMINANCE16_ALPHA16 = 32840;
const long GL_INTENSITY = 32841;
const long GL_INTENSITY4 = 32842;
const long GL_INTENSITY8 = 32843;
const long GL_INTENSITY12 = 32844;
const long GL_INTENSITY16 = 32845;
const long GL_R3_G3_B2 = 10768;
const long GL_RGB4 = 32847;
const long GL_RGB5 = 32848;
const long GL_RGB8 = 32849;
const long GL_RGB10 = 32850;
const long GL_RGB12 = 32851;
const long GL_RGB16 = 32852;
const long GL_RGBA2 = 32853;
const long GL_RGBA4 = 32854;
const long GL_RGB5_A1 = 32855;
const long GL_RGBA8 = 32856;
const long GL_RGB10_A2 = 32857;
const long GL_RGBA12 = 32858;
const long GL_RGBA16 = 32859;
const long GL_TEXTURE_RED_SIZE = 32860;
const long GL_TEXTURE_GREEN_SIZE = 32861;
const long GL_TEXTURE_BLUE_SIZE = 32862;
```

```
const long GL_TEXTURE_ALPHA_SIZE = 32863;
const long GL_TEXTURE_LUMINANCE_SIZE = 32864;
const long GL_TEXTURE_INTENSITY_SIZE = 32865;
const long GL_PROXY_TEXTURE_1D = 32867;
const long GL_PROXY_TEXTURE_2D = 32868;
const long GL_TEXTURE_PRIORITY = 32870;
const long GL_TEXTURE_RESIDENT = 32871;
const long GL_TEXTURE_BINDING_1D = 32872;

const long GL_TEXTURE_BINDING_2D = 32873;
const long GL_VERTEX_ARRAY = 32884;
const long GL_NORMAL_ARRAY = 32885;
const long GL_COLOR_ARRAY = 32886;
const long GL_INDEX_ARRAY = 32887;
const long GL_TEXTURE_COORD_ARRAY = 32888;
const long GL_EDGE_FLAG_ARRAY = 32889;
const long GL_VERTEX_ARRAY_SIZE = 32890;
const long GL_VERTEX_ARRAY_TYPE = 32891;
const long GL_VERTEX_ARRAY_STRIDE = 32892;
const long GL_NORMAL_ARRAY_TYPE = 32894;
const long GL_NORMAL_ARRAY_STRIDE = 32895;
const long GL_COLOR_ARRAY_SIZE = 32897;
const long GL_COLOR_ARRAY_TYPE = 32898;
const long GL_COLOR_ARRAY_STRIDE = 32899;
const long GL_INDEX_ARRAY_TYPE = 32901;
const long GL_INDEX_ARRAY_STRIDE = 32902;
const long GL_TEXTURE_COORD_ARRAY_SIZE = 32904;
const long GL_TEXTURE_COORD_ARRAY_TYPE = 32905;
const long GL_TEXTURE_COORD_ARRAY_STRIDE = 32906;
const long GL_EDGE_FLAG_ARRAY_STRIDE = 32908;
const long GL_VERTEX_ARRAY_POINTER = 32910;
const long GL_NORMAL_ARRAY_POINTER = 32911;
const long GL_COLOR_ARRAY_POINTER = 32912;
const long GL_INDEX_ARRAY_POINTER = 32913;
const long GL_TEXTURE_COORD_ARRAY_POINTER = 32914;
const long GL_EDGE_FLAG_ARRAY_POINTER = 32915;
const long GL_V2F = 10784;
const long GL_V3F = 10785;
const long GL_C4UB_V2F = 10786;
const long GL_C4UB_V3F = 10787;
const long GL_C3F_V3F = 10788;
const long GL_N3F_V3F = 10789;
const long GL_C4F_N3F_V3F = 10790;
const long GL_T2F_V3F = 10791;
const long GL_T4F_V4F = 10792;
const long GL_T2F_C4UB_V3F = 10793;
const long GL_T2F_C3F_V3F = 10794;
const long GL_T2F_N3F_V3F = 10795;
const long GL_T2F_C4F_N3F_V3F = 10796;
const long GL_T4F_C4F_N3F_V4F = 10797;
const long GL_EXT_vertex_array = 1;
const long GL_WIN_swap_hint = 1;
const long GL_EXT_bgra = 1;
const long GL_EXT_paletted_texture = 1;
const long GL_VERTEX_ARRAY_EXT = 32884;
const long GL_NORMAL_ARRAY_EXT = 32885;
const long GL_COLOR_ARRAY_EXT = 32886;
const long GL_INDEX_ARRAY_EXT = 32887;
```

```
const long GL_TEXTURE_COORD_ARRAY_EXT = 32888;
const long GL_EDGE_FLAG_ARRAY_EXT = 32889;
const long GL_VERTEX_ARRAY_SIZE_EXT = 32890;
const long GL_VERTEX_ARRAY_TYPE_EXT = 32891;
const long GL_VERTEX_ARRAY_STRIDE_EXT = 32892;
const long GL_VERTEX_ARRAY_COUNT_EXT = 32893;
const long GL_NORMAL_ARRAY_TYPE_EXT = 32894;
const long GL_NORMAL_ARRAY_STRIDE_EXT = 32895;
const long GL_NORMAL_ARRAY_COUNT_EXT = 32896;
const long GL_COLOR_ARRAY_SIZE_EXT = 32897;
const long GL_COLOR_ARRAY_TYPE_EXT = 32898;
const long GL_COLOR_ARRAY_STRIDE_EXT = 32899;
const long GL_COLOR_ARRAY_COUNT_EXT = 32900;
const long GL_INDEX_ARRAY_TYPE_EXT = 32901;
const long GL_INDEX_ARRAY_STRIDE_EXT = 32902;
const long GL_INDEX_ARRAY_COUNT_EXT = 32903;
const long GL_TEXTURE_COORD_ARRAY_SIZE_EXT = 32904;
const long GL_TEXTURE_COORD_ARRAY_TYPE_EXT = 32905;
const long GL_TEXTURE_COORD_ARRAY_STRIDE_EXT = 32906;
const long GL_TEXTURE_COORD_ARRAY_COUNT_EXT = 32907;
const long GL_EDGE_FLAG_ARRAY_STRIDE_EXT = 32908;
const long GL_EDGE_FLAG_ARRAY_COUNT_EXT = 32909;
const long GL_VERTEX_ARRAY_POINTER_EXT = 32910;
const long GL_NORMAL_ARRAY_POINTER_EXT = 32911;
const long GL_COLOR_ARRAY_POINTER_EXT = 32912;
const long GL_INDEX_ARRAY_POINTER_EXT = 32913;
const long GL_TEXTURE_COORD_ARRAY_POINTER_EXT = 32914;
const long GL_EDGE_FLAG_ARRAY_POINTER_EXT = 32915;
const long GL_BGR_EXT = 32992;
const long GL_BGRA_EXT = 32993;
const long GL_COLOR_TABLE_FORMAT_EXT = 32984;
const long GL_COLOR_TABLE_WIDTH_EXT = 32985;
const long GL_COLOR_TABLE_RED_SIZE_EXT = 32986;
const long GL_COLOR_TABLE_GREEN_SIZE_EXT = 32987;
const long GL_COLOR_TABLE_BLUE_SIZE_EXT = 32988;
const long GL_COLOR_TABLE_ALPHA_SIZE_EXT = 32989;
const long GL_COLOR_TABLE_LUMINANCE_SIZE_EXT = 32990;
const long GL_COLOR_TABLE_INTENSITY_SIZE_EXT = 32991;
const long GL_COLOR_INDEX1_EXT = 32994;
const long GL_COLOR_INDEX2_EXT = 32995;
const long GL_COLOR_INDEX4_EXT = 32996;
const long GL_COLOR_INDEX8_EXT = 32997;
const long GL_COLOR_INDEX12_EXT = 32998;
const long GL_COLOR_INDEX16_EXT = 32999;
[entry("glAccum")]
void _stdcall glAccum([in] int op, [in] float value );
[entry("glAlphaFunc")]
void _stdcall glAlphaFunc([in] int func, [in] float ref );
[entry("glAreTexturesResident")]
unsigned char _stdcall glAreTexturesResident([in] int n, [in, out] int * textures,
    [in, out] unsigned char * residences );
[entry("glArrayElement")]
void _stdcall glArrayElement([in] int i );
[entry("glBegin")]
void _stdcall glBegin([in] int mode );
[entry("glBindTexture")]
void _stdcall glBindTexture([in] int target, [in] int texture );
[entry("glBitmap")]
```

```
void _stdcall glBitmap([in] int width, [in] int height, [in] float xorig, [in] float yorig,
                [in] float xmove, [in] float ymove, [in, out] unsigned char * bitmap );
[entry("glBlendFunc")]
void _stdcall glBlendFunc([in] int sfactor, [in] int dfactor );
[entry("glCallList")]
void _stdcall glCallList([in] int list );
[entry("glCallLists")]
void _stdcall glCallLists([in] int n, [in] int type, [in, out] void * lists );
[entry("glClear")]
void _stdcall glClear([in] int mask );
[entry("glClearAccum")]
void _stdcall glClearAccum([in] float red, [in] float green, [in] float blue, [in] float alpha );
[entry("glClearColor")]
void _stdcall glClearColor([in] float red, [in] float green, [in] float blue, [in] float alpha );
[entry("glClearDepth")]
void _stdcall glClearDepth([in] double depth );
[entry("glClearIndex")]
void _stdcall glClearIndex([in] float c );
[entry("glClearStencil")]
void _stdcall glClearStencil([in] int s );
[entry("glClipPlane")]
void _stdcall glClipPlane([in] int plane, [in, out] double * equation );
[entry("glColor3b")]
void _stdcall glColor3b([in] unsigned char red, [in] unsigned char green, [in] unsigned char blue );
[entry("glColor3bv")]
void _stdcall glColor3bv([in, out] unsigned char * v );
[entry("glColor3d")]
void _stdcall glColor3d([in] double red, [in] double green, [in] double blue );
[entry("glColor3dv")]
void _stdcall glColor3dv([in, out] double * v );
[entry("glColor3f")]
void _stdcall glColor3f([in] float red, [in] float green, [in] float blue );
[entry("glColor3fv")]
void _stdcall glColor3fv([in, out] float * v );
[entry("glColor3i")]
void _stdcall glColor3i([in] int red, [in] int green, [in] int blue );
[entry("glColor3iv")]
void _stdcall glColor3iv([in, out] int * v );
[entry("glColor3s")]
void _stdcall glColor3s([in] short red, [in] short green, [in] short blue );
[entry("glColor3sv")]
void _stdcall glColor3sv([in, out] short * v );
[entry("glColor3ub")]
void _stdcall glColor3ub([in] unsigned char red, [in] unsigned char green, [in] unsigned char blue );
[entry("glColor3ubv")]
void _stdcall glColor3ubv([in, out] unsigned char * v );
[entry("glColor3ui")]
void _stdcall glColor3ui([in] int red, [in] int green, [in] int blue );
[entry("glColor3uiv")]
void _stdcall glColor3uiv([in, out] int * v );
[entry("glColor3us")]
void _stdcall glColor3us([in] short red, [in] short green, [in] short blue );
[entry("glColor3usv")]
void _stdcall glColor3usv([in, out] short * v );
[entry("glColor4b")]
void _stdcall glColor4b([in] unsigned char red, [in] unsigned char green,
       [in] unsigned char blue, [in] unsigned char alpha );
[entry("glColor4bv")]
```

```
void _stdcall glColor4bv([in, out] unsigned char * v );
[entry("glColor4d")]
void _stdcall glColor4d([in] double red, [in] double green, [in] double blue, [in] double alpha );
[entry("glColor4dv")]
void _stdcall glColor4dv([in, out] double * v );
[entry("glColor4f")]
void _stdcall glColor4f([in] float red, [in] float green, [in] float blue, [in] float alpha );
[entry("glColor4fv")]
void _stdcall glColor4fv([in, out] float * v );
[entry("glColor4i")]
void _stdcall glColor4i([in] int red, [in] int green, [in] int blue, [in] int alpha );
[entry("glColor4iv")]
void _stdcall glColor4iv([in, out] int * v );
[entry("glColor4s")]
void _stdcall glColor4s([in] short red, [in] short green, [in] short blue, [in] short alpha );
[entry("glColor4sv")]
void _stdcall glColor4sv([in, out] short * v );
[entry("glColor4ub")]
void _stdcall glColor4ub([in] unsigned char red, [in] unsigned char green,
        [in] unsigned char blue, [in] unsigned char alpha );
[entry("glColor4ubv")]
void _stdcall glColor4ubv([in, out] unsigned char * v );
[entry("glColor4ui")]
void _stdcall glColor4ui([in] int red, [in] int green, [in] int blue, [in] int alpha );
[entry("glColor4uiv")]
void _stdcall glColor4uiv([in, out] int * v );
[entry("glColor4us")]
void _stdcall glColor4us([in] short red, [in] short green, [in] short blue, [in] short alpha );
[entry("glColor4usv")]
void _stdcall glColor4usv([in, out] short * v );
[entry("glColorMask")]
void _stdcall glColorMask([in] unsigned char red, [in] unsigned char green,
            [in] unsigned char blue, [in] unsigned char alpha );
[entry("glColorMaterial")]
void _stdcall glColorMaterial([in] int face, [in] int mode );
[entry("glColorPointer")]
void _stdcall glColorPointer([in] int size, [in] int type, [in] int stride, [in, out] void * pointer );
[entry("glCopyPixels")]
void _stdcall glCopyPixels([in] int x, [in] int y, [in] int width, [in] int height, [in] int type );
[entry("glCopyTexImage1D")]
void _stdcall glCopyTexImage1D([in] int target, [in] int level, [in] int internalFormat, [in] int x,
      [in] int y, [in] int width, [in] int border );
[entry("glCopyTexImage2D")]
void _stdcall glCopyTexImage2D([in] int target, [in] int level, [in] int internalFormat, [in] int x,
     [in] int y, [in] int width, [in] int height, [in] int border );
[entry("glCopyTexSubImage1D")]
void _stdcall glCopyTexSubImage1D([in] int target, [in] int level, [in] int xoffset, [in] int x,
      [in] int y, [in] int width );
[entry("glCopyTexSubImage2D")]
void _stdcall glCopyTexSubImage2D([in] int target, [in] int level, [in] int xoffset, [in] int yoffset,
     [in] int x, [in] int y, [in] int width, [in] int height );
[entry("glCullFace")]
void _stdcall glCullFace([in] int mode );
[entry("glDeleteLists")]
void _stdcall glDeleteLists([in] int list, [in] int range );
[entry("glDeleteTextures")]
void _stdcall glDeleteTextures([in] int n, [in, out] int * textures );
[entry("glDepthFunc")]
```

```
void _stdcall glDepthFunc([in] int func );
[entry("glDepthMask")]
void _stdcall glDepthMask([in] unsigned char flag );
[entry("glDepthRange")]
void _stdcall glDepthRange([in] double zNear, [in] double zFar );
[entry("glDisable")]
void _stdcall glDisable([in] int cap );
[entry("glDisableClientState")]
void _stdcall glDisableClientState([in] int array );
[entry("glDrawArrays")]
void _stdcall glDrawArrays([in] int mode, [in] int first, [in] int count );
[entry("glDrawBuffer")]
void _stdcall glDrawBuffer([in] int mode );
[entry("glDrawElements")]
void _stdcall glDrawElements([in] int mode, [in] int count, [in] int type, [in, out] void * indices );
[entry("glDrawPixels")]
void _stdcall glDrawPixels([in] int width, [in] int height, [in] int format, [in] int type,
    [in, out] void * pixels );
[entry("glEdgeFlag")]
void _stdcall glEdgeFlag([in] unsigned char flag );
[entry("glEdgeFlagPointer")]
void _stdcall glEdgeFlagPointer([in] int stride, [in, out] void * pointer );
[entry("glEdgeFlagv")]
void _stdcall glEdgeFlagv([in, out] unsigned char * flag );
[entry("glEnable")]
void _stdcall glEnable([in] int cap );
[entry("glEnableClientState")]
void _stdcall glEnableClientState([in] int array );
[entry("glEnd")]
void _stdcall glEnd( void );
[entry("glEndList")]
void _stdcall glEndList( void );
[entry("glEvalCoord1d")]
void _stdcall glEvalCoord1d([in] double u );
[entry("glEvalCoord1dv")]
void _stdcall glEvalCoord1dv([in, out] double * u );
[entry("glEvalCoord1f")]
void _stdcall glEvalCoord1f([in] float u );
[entry("glEvalCoord1fv")]
void _stdcall glEvalCoord1fv([in, out] float * u );
[entry("glEvalCoord2d")]
void _stdcall glEvalCoord2d([in] double u, [in] double v );
[entry("glEvalCoord2dv")]
void _stdcall glEvalCoord2dv([in, out] double * u );
[entry("glEvalCoord2f")]
void _stdcall glEvalCoord2f([in] float u, [in] float v );
[entry("glEvalCoord2fv")]
void _stdcall glEvalCoord2fv([in, out] float * u );
[entry("glEvalMesh1")]
void _stdcall glEvalMesh1([in] int mode, [in] int i1, [in] int i2 );
[entry("glEvalMesh2")]
void _stdcall glEvalMesh2([in] int mode, [in] int i1, [in] int i2, [in] int j1, [in] int j2 );
[entry("glEvalPoint1")]
void _stdcall glEvalPoint1([in] int i );
[entry("glEvalPoint2")]
void _stdcall glEvalPoint2([in] int i, [in] int j );
[entry("glFeedbackBuffer")]
void _stdcall glFeedbackBuffer([in] int size, [in] int type, [in, out] float * buffer );
```

```
[entry("glFinish")]
void _stdcall glFinish( void );
[entry("glFlush")]
void _stdcall glFlush( void );
[entry("glFogf")]
void _stdcall glFogf([in] int pname, [in] float param );
[entry("glFogfv")]
void _stdcall glFogfv([in] int pname, [in, out] float * params );
[entry("glFogi")]
void _stdcall glFogi([in] int pname, [in] int param );
[entry("glFogiv")]
void _stdcall glFogiv([in] int pname, [in, out] int * params );
[entry("glFrontFace")]
void _stdcall glFrontFace([in] int mode );
[entry("glFrustum")]
void _stdcall glFrustum([in] double left, [in] double right, [in] double bottom,
      [in] double top, [in] double zNear, [in] double zFar );
[entry("glGenLists")]
int _stdcall glGenLists([in] int range );
[entry("glGenTextures")]
void _stdcall glGenTextures([in] int n, [in, out] int * textures );
[entry("glGetBooleanv")]
void _stdcall glGetBooleanv([in] int pname, [in, out] unsigned char * params );
[entry("glGetClipPlane")]
void _stdcall glGetClipPlane([in] int plane, [in, out] double * equation );
[entry("glGetDoublev")]
void _stdcall glGetDoublev([in] int pname, [in, out] double * params );
[entry("glGetError")]
int _stdcall glGetError( void );
[entry("glGetFloatv")]
void _stdcall glGetFloatv([in] int pname, [in, out] float * params );
[entry("glGetIntegerv")]
void _stdcall glGetIntegerv([in] int pname, [in, out] int * params );
[entry("glGetLightfv")]
void _stdcall glGetLightfv([in] int light, [in] int pname, [in, out] float * params );
[entry("glGetLightiv")]
void _stdcall glGetLightiv([in] int light, [in] int pname, [in, out] int * params );
[entry("glGetMapdv")]
void _stdcall glGetMapdv([in] int target, [in] int query, [in, out] double * v );
[entry("glGetMapfv")]
void _stdcall glGetMapfv([in] int target, [in] int query, [in, out] float * v );
[entry("glGetMapiv")]
void _stdcall glGetMapiv([in] int target, [in] int query, [in, out] int * v );
[entry("glGetMaterialfv")]
void _stdcall glGetMaterialfv([in] int face, [in] int pname, [in, out] float * params );
[entry("glGetMaterialiv")]
void _stdcall glGetMaterialiv([in] int face, [in] int pname, [in, out] int * params );
[entry("glGetPixelMapfv")]
void _stdcall glGetPixelMapfv([in] int map, [in, out] float * values );
[entry("glGetPixelMapuiv")]
void _stdcall glGetPixelMapuiv([in] int map, [in, out] int * values );
[entry("glGetPixelMapusv")]
void _stdcall glGetPixelMapusv([in] int map, [in, out] short * values );
[entry("glGetPointerv")]
void _stdcall glGetPointerv([in] int pname, [in, out] void ** params );
[entry("glGetPolygonStipple")]
void _stdcall glGetPolygonStipple([in, out] unsigned char * mask );
[entry("glGetString")]
```

```
unsigned char * _stdcall glGetString([in] int name );
[entry("glGetTexEnvfv")]
void _stdcall glGetTexEnvfv([in] int target, [in] int pname, [in, out] float * params );
[entry("glGetTexEnviv")]
void _stdcall glGetTexEnviv([in] int target, [in] int pname, [in, out] int * params );
[entry("glGetTexGendv")]
void _stdcall glGetTexGendv([in] int coord, [in] int pname, [in, out] double * params );
[entry("glGetTexGenfv")]
void _stdcall glGetTexGenfv([in] int coord, [in] int pname, [in, out] float * params );
[entry("glGetTexGeniv")]
void _stdcall glGetTexGeniv([in] int coord, [in] int pname, [in, out] int * params );
[entry("glGetTexImage")]
void _stdcall glGetTexImage([in] int target, [in] int level, [in] int format, [in] int type,
    [in, out] void * pixels );
[entry("glGetTexLevelParameterfv")]
void _stdcall glGetTexLevelParameterfv([in] int target, [in] int level, [in] int pname,
    [in, out] float * params );
[entry("glGetTexLevelParameteriv")]
void _stdcall glGetTexLevelParameteriv([in] int target, [in] int level, [in] int pname,
    [in, out] int * params );
[entry("glGetTexParameterfv")]
void _stdcall glGetTexParameterfv([in] int target, [in] int pname, [in, out] float * params );
[entry("glGetTexParameteriv")]
void _stdcall glGetTexParameteriv([in] int target, [in] int pname, [in, out] int * params );
[entry("glHint")]
void _stdcall glHint([in] int target, [in] int mode );
[entry("glIndexMask")]
void _stdcall glIndexMask([in] int mask );
[entry("glIndexPointer")]
void _stdcall glIndexPointer([in] int type, [in] int stride, [in, out] void * pointer );
[entry("glIndexd")]
void _stdcall glIndexd([in] double c );
[entry("glIndexdv")]
void _stdcall glIndexdv([in, out] double * c );
[entry("glIndexf")]
void _stdcall glIndexf([in] float c );
[entry("glIndexfv")]
void _stdcall glIndexfv([in, out] float * c );
[entry("glIndexi")]
void _stdcall glIndexi([in] int c );
[entry("glIndexiv")]
void _stdcall glIndexiv([in, out] int * c );
[entry("glIndexs")]
void _stdcall glIndexs([in] short c );
[entry("glIndexsv")]
void _stdcall glIndexsv([in, out] short * c );
[entry("glIndexub")]
void _stdcall glIndexub([in] unsigned char c );
[entry("glIndexubv")]
void _stdcall glIndexubv([in, out] unsigned char * c );
[entry("glInitNames")]
void _stdcall glInitNames( void );
[entry("glInterleavedArrays")]
void _stdcall glInterleavedArrays([in] int format, [in] int stride, [in, out] void * pointer );
[entry("glIsEnabled")]
unsigned char _stdcall glIsEnabled([in] int cap );
[entry("glIsList")]
unsigned char _stdcall glIsList([in] int list );
```

```
[entry("glIsTexture")]
unsigned char _stdcall glIsTexture([in] int texture );
[entry("glLightModelf")]
void _stdcall glLightModelf([in] int pname, [in] float param );
[entry("glLightModelfv")]
void _stdcall glLightModelfv([in] int pname, [in, out] float * params );
[entry("glLightModeli")]
void _stdcall glLightModeli([in] int pname, [in] int param );
[entry("glLightModeliv")]
void _stdcall glLightModeliv([in] int pname, [in, out] int * params );
[entry("glLightf")]
void _stdcall glLightf([in] int light, [in] int pname, [in] float param );
[entry("glLightfv")]
void _stdcall glLightfv([in] int light, [in] int pname, [in, out] float * params );
[entry("glLighti")]
void _stdcall glLighti([in] int light, [in] int pname, [in] int param );
[entry("glLightiv")]
void _stdcall glLightiv([in] int light, [in] int pname, [in, out] int * params );
[entry("glLineStipple")]
void _stdcall glLineStipple([in] int factor, [in] short pattern );
[entry("glLineWidth")]
void _stdcall glLineWidth([in] float width );
[entry("glListBase")]
void _stdcall glListBase([in] int base );
[entry("glLoadIdentity")]
void _stdcall glLoadIdentity( void );
[entry("glLoadMatrixd")]
void _stdcall glLoadMatrixd([in, out] double * m );
[entry("glLoadMatrixf")]
void _stdcall glLoadMatrixf([in, out] float * m );
[entry("glLoadName")]
void _stdcall glLoadName([in] int name );
[entry("glLogicOp")]
void _stdcall glLogicOp([in] int opcode );
[entry("glMap1d")]
void _stdcall glMap1d([in] int target, [in] double u1, [in] double u2, [in] int stride,
[in] int order, [in, out] double * points );
[entry("glMap1f")]
void _stdcall glMap1f([in] int target, [in] float u1, [in] float u2, [in] int stride,
 [in] int order, [in, out] float * points );
[entry("glMap2d")]
void _stdcall glMap2d([in] int target, [in] double u1, [in] double u2, [in] int ustride,
  [in] int uorder, [in] double v1, [in] double v2, [in] int vstride,
  [in] int vorder, [in, out] double * points );
[entry("glMap2f")]
void _stdcall glMap2f([in] int target, [in] float u1, [in] float u2, [in] int ustride,
[in] int uorder, [in] float v1, [in] float v2, [in] int vstride, [in] int vorder, [in, out] float * points );
[entry("glMapGrid1d")]
void _stdcall glMapGrid1d([in] int un, [in] double u1, [in] double u2 );
[entry("glMapGrid1f")]
void _stdcall glMapGrid1f([in] int un, [in] float u1, [in] float u2 );
[entry("glMapGrid2d")]
void _stdcall glMapGrid2d([in] int un, [in] double u1, [in] double u2, [in] int vn,
    [in] double v1, [in] double v2 );
[entry("glMapGrid2f")]
void _stdcall glMapGrid2f([in] int un, [in] float u1, [in] float u2, [in] int vn,
   [in] float v1, [in] float v2 );
[entry("glMaterialf")]
```

```
void _stdcall glMaterialf([in] int face, [in] int pname, [in] float param );
[entry("glMaterialfv")]
void _stdcall glMaterialfv([in] int face, [in] int pname, [in, out] float * params );
[entry("glMateriali")]
void _stdcall glMateriali([in] int face, [in] int pname, [in] int param );
[entry("glMaterialiv")]
void _stdcall glMaterialiv([in] int face, [in] int pname, [in, out] int * params );
[entry("glMatrixMode")]
void _stdcall glMatrixMode([in] int mode );
[entry("glMultMatrixd")]
void _stdcall glMultMatrixd([in, out] double * m );
[entry("glMultMatrixf")]
void _stdcall glMultMatrixf([in, out] float * m );
[entry("glNewList")]
void _stdcall glNewList([in] int list, [in] int mode );
[entry("glNormal3b")]
void _stdcall glNormal3b([in] unsigned char nx, [in] unsigned char ny, [in] unsigned char nz );
[entry("glNormal3bv")]
void _stdcall glNormal3bv([in, out] unsigned char * v );
[entry("glNormal3d")]
void _stdcall glNormal3d([in] double nx, [in] double ny, [in] double nz );
[entry("glNormal3dv")]
void _stdcall glNormal3dv([in, out] double * v );
[entry("glNormal3f")]
void _stdcall glNormal3f([in] float nx, [in] float ny, [in] float nz );
[entry("glNormal3fv")]
void _stdcall glNormal3fv([in, out] float * v );
[entry("glNormal3i")]
void _stdcall glNormal3i([in] int nx, [in] int ny, [in] int nz );
[entry("glNormal3iv")]
void _stdcall glNormal3iv([in, out] int * v );
[entry("glNormal3s")]
void _stdcall glNormal3s([in] short nx, [in] short ny, [in] short nz );
[entry("glNormal3sv")]
void _stdcall glNormal3sv([in, out] short * v );

[entry("glNormalPointer")]
void _stdcall glNormalPointer([in] int type, [in] int stride, [in, out] void * pointer );
[entry("glOrtho")]
void _stdcall glOrtho([in] double left, [in] double right, [in] double bottom,
    [in] double top, [in] double zNear, [in] double zFar );
[entry("glPassThrough")]
void _stdcall glPassThrough([in] float token );
[entry("glPixelMapfv")]
void _stdcall glPixelMapfv([in] int map, [in] int mapsize, [in, out] float * values );
[entry("glPixelMapuiv")]
void _stdcall glPixelMapuiv([in] int map, [in] int mapsize, [in, out] int * values );
[entry("glPixelMapusv")]
void _stdcall glPixelMapusv([in] int map, [in] int mapsize, [in, out] short * values );
[entry("glPixelStoref")]
void _stdcall glPixelStoref([in] int pname, [in] float param );
[entry("glPixelStorei")]
void _stdcall glPixelStorei([in] int pname, [in] int param );
[entry("glPixelTransferf")]
void _stdcall glPixelTransferf([in] int pname, [in] float param );
[entry("glPixelTransferi")]
void _stdcall glPixelTransferi([in] int pname, [in] int param );
[entry("glPixelZoom")]
```

```
void _stdcall glPixelZoom([in] float xfactor, [in] float yfactor );
[entry("glPointSize")]
void _stdcall glPointSize([in] float size );
[entry("glPolygonMode")]
void _stdcall glPolygonMode([in] int face, [in] int mode );
[entry("glPolygonOffset")]
void _stdcall glPolygonOffset([in] float factor, [in] float units );
[entry("glPolygonStipple")]
void _stdcall glPolygonStipple([in, out] unsigned char * mask );
[entry("glPopAttrib")]
void _stdcall glPopAttrib( void );
[entry("glPopClientAttrib")]
void _stdcall glPopClientAttrib( void );
[entry("glPopMatrix")]
void _stdcall glPopMatrix( void );
[entry("glPopName")]
void _stdcall glPopName( void );
[entry("glPrioritizeTextures")]
void _stdcall glPrioritizeTextures([in] int n, [in, out] int * textures, [in, out] float * priorities );
[entry("glPushAttrib")]
void _stdcall glPushAttrib([in] int mask );
[entry("glPushClientAttrib")]
void _stdcall glPushClientAttrib([in] int mask );
[entry("glPushMatrix")]
void _stdcall glPushMatrix( void );
[entry("glPushName")]
void _stdcall glPushName([in] int name );
[entry("glRasterPos2d")]
void _stdcall glRasterPos2d([in] double x, [in] double y );
[entry("glRasterPos2dv")]
void _stdcall glRasterPos2dv([in, out] double * v );
[entry("glRasterPos2f")]
void _stdcall glRasterPos2f([in] float x, [in] float y );
[entry("glRasterPos2fv")]
void _stdcall glRasterPos2fv([in, out] float * v );
[entry("glRasterPos2i")]
void _stdcall glRasterPos2i([in] int x, [in] int y );
[entry("glRasterPos2iv")]
void _stdcall glRasterPos2iv([in, out] int * v );
[entry("glRasterPos2s")]
void _stdcall glRasterPos2s([in] short x, [in] short y );
[entry("glRasterPos2sv")]
void _stdcall glRasterPos2sv([in, out] short * v );
[entry("glRasterPos3d")]
void _stdcall glRasterPos3d([in] double x, [in] double y, [in] double z );
[entry("glRasterPos3dv")]
void _stdcall glRasterPos3dv([in, out] double * v );
[entry("glRasterPos3f")]
void _stdcall glRasterPos3f([in] float x, [in] float y, [in] float z );
[entry("glRasterPos3fv")]
void _stdcall glRasterPos3fv([in, out] float * v );
[entry("glRasterPos3i")]
void _stdcall glRasterPos3i([in] int x, [in] int y, [in] int z );
[entry("glRasterPos3iv")]
void _stdcall glRasterPos3iv([in, out] int * v );
[entry("glRasterPos3s")]
void _stdcall glRasterPos3s([in] short x, [in] short y, [in] short z );
[entry("glRasterPos3sv")]
```

```
void _stdcall glRasterPos3sv([in, out] short * v );
[entry("glRasterPos4d")]
void _stdcall glRasterPos4d([in] double x, [in] double y, [in] double z, [in] double w );
[entry("glRasterPos4dv")]
void _stdcall glRasterPos4dv([in, out] double * v );
[entry("glRasterPos4f")]
void _stdcall glRasterPos4f([in] float x, [in] float y, [in] float z, [in] float w );
[entry("glRasterPos4fv")]
void _stdcall glRasterPos4fv([in, out] float * v );
[entry("glRasterPos4i")]
void _stdcall glRasterPos4i([in] int x, [in] int y, [in] int z, [in] int w );
[entry("glRasterPos4iv")]
void _stdcall glRasterPos4iv([in, out] int * v );
[entry("glRasterPos4s")]
void _stdcall glRasterPos4s([in] short x, [in] short y, [in] short z, [in] short w );
[entry("glRasterPos4sv")]
void _stdcall glRasterPos4sv([in, out] short * v );
[entry("glReadBuffer")]
void _stdcall glReadBuffer([in] int mode );
[entry("glReadPixels")]
void _stdcall glReadPixels([in] int x, [in] int y, [in] int width, [in] int height,
    [in] int format, [in] int type, [in, out] void * pixels );
[entry("glRectd")]
void _stdcall glRectd([in] double x1, [in] double y1, [in] double x2, [in] double y2 );
[entry("glRectdv")]
void _stdcall glRectdv([in, out] double * v1, [in, out] double * v2 );
[entry("glRectf")]
void _stdcall glRectf([in] float x1, [in] float y1, [in] float x2, [in] float y2 );
[entry("glRectfv")]
void _stdcall glRectfv([in, out] float * v1, [in, out] float * v2 );
[entry("glRecti")]
void _stdcall glRecti([in] int x1, [in] int y1, [in] int x2, [in] int y2 );
[entry("glRectiv")]
void _stdcall glRectiv([in, out] int * v1, [in, out] int * v2 );
[entry("glRects")]
void _stdcall glRects([in] short x1, [in] short y1, [in] short x2, [in] short y2 );
[entry("glRectsv")]
void _stdcall glRectsv([in, out] short * v1, [in, out] short * v2 );
[entry("glRenderMode")]
int _stdcall glRenderMode([in] int mode );
[entry("glRotated")]
void _stdcall glRotated([in] double angle, [in] double x, [in] double y, [in] double z );
[entry("glRotatef")]
void _stdcall glRotatef([in] float angle, [in] float x, [in] float y, [in] float z );
[entry("glScaled")]
void _stdcall glScaled([in] double x, [in] double y, [in] double z );
[entry("glScalef")]
void _stdcall glScalef([in] float x, [in] float y, [in] float z );
[entry("glScissor")]
void _stdcall glScissor([in] int x, [in] int y, [in] int width, [in] int height );
[entry("glSelectBuffer")]
void _stdcall glSelectBuffer([in] int size, [in, out] int * buffer );
[entry("glShadeModel")]
void _stdcall glShadeModel([in] int mode );
[entry("glStencilFunc")]
void _stdcall glStencilFunc([in] int func, [in] int ref, [in] int mask );
[entry("glStencilMask")]
void _stdcall glStencilMask([in] int mask );
```

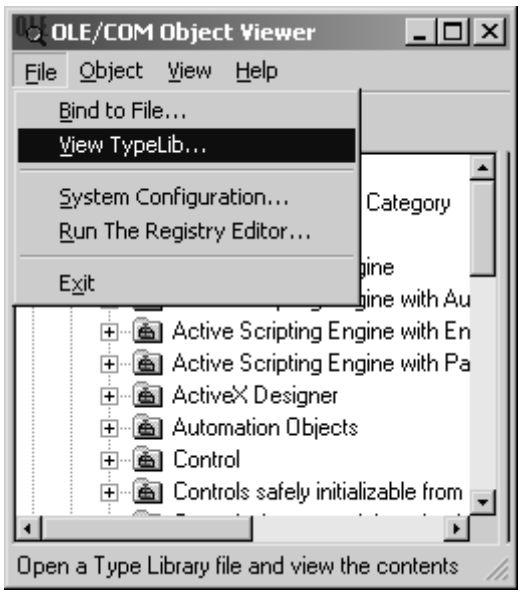
```
[entry("glStencilOp")]
void _stdcall glStencilOp([in] int fail, [in] int zfail, [in] int zpass );
[entry("glTexCoord1d")]
void _stdcall glTexCoord1d([in] double s );
[entry("glTexCoord1dv")]
void _stdcall glTexCoord1dv([in, out] double * v );
[entry("glTexCoord1f")]
void _stdcall glTexCoord1f([in] float s );
[entry("glTexCoord1fv")]
void _stdcall glTexCoord1fv([in, out] float * v );
[entry("glTexCoord1i")]
void _stdcall glTexCoord1i([in] int s );
[entry("glTexCoord1iv")]
void _stdcall glTexCoord1iv([in, out] int * v );
[entry("glTexCoord1s")]
void _stdcall glTexCoord1s([in] short s );
[entry("glTexCoord1sv")]
void _stdcall glTexCoord1sv([in, out] short * v );
[entry("glTexCoord2d")]
void _stdcall glTexCoord2d([in] double s, [in] double t );
[entry("glTexCoord2dv")]
void _stdcall glTexCoord2dv([in, out] double * v );
[entry("glTexCoord2f")]
void _stdcall glTexCoord2f([in] float s, [in] float t );
[entry("glTexCoord2fv")]
void _stdcall glTexCoord2fv([in, out] float * v );
[entry("glTexCoord2i")]
void _stdcall glTexCoord2i([in] int s, [in] int t );
[entry("glTexCoord2iv")]
void _stdcall glTexCoord2iv([in, out] int * v );
[entry("glTexCoord2s")]
void _stdcall glTexCoord2s([in] short s, [in] short t );
[entry("glTexCoord2sv")]
void _stdcall glTexCoord2sv([in, out] short * v );
[entry("glTexCoord3d")]
void _stdcall glTexCoord3d([in] double s, [in] double t, [in] double r );
[entry("glTexCoord3dv")]
void _stdcall glTexCoord3dv([in, out] double * v );
[entry("glTexCoord3f")]
void _stdcall glTexCoord3f([in] float s, [in] float t, [in] float r );
[entry("glTexCoord3fv")]
void _stdcall glTexCoord3fv([in, out] float * v );
[entry("glTexCoord3i")]
void _stdcall glTexCoord3i([in] int s, [in] int t, [in] int r );
[entry("glTexCoord3iv")]
void _stdcall glTexCoord3iv([in, out] int * v );
[entry("glTexCoord3s")]
void _stdcall glTexCoord3s([in] short s, [in] short t, [in] short r );
[entry("glTexCoord3sv")]
void _stdcall glTexCoord3sv([in, out] short * v );
[entry("glTexCoord4d")]
void _stdcall glTexCoord4d([in] double s, [in] double t, [in] double r, [in] double q );
[entry("glTexCoord4dv")]
void _stdcall glTexCoord4dv([in, out] double * v );
[entry("glTexCoord4f")]
void _stdcall glTexCoord4f([in] float s, [in] float t, [in] float r, [in] float q );
[entry("glTexCoord4fv")]
void _stdcall glTexCoord4fv([in, out] float * v );
```

```
[entry("glTexCoord4i")]
void _stdcall glTexCoord4i([in] int s, [in] int t, [in] int r, [in] int q );
[entry("glTexCoord4iv")]
void _stdcall glTexCoord4iv([in, out] int * v );
[entry("glTexCoord4s")]
void _stdcall glTexCoord4s([in] short s, [in] short t, [in] short r, [in] short q );
[entry("glTexCoord4sv")]
void _stdcall glTexCoord4sv([in, out] short * v );
[entry("glTexCoordPointer")]
void _stdcall glTexCoordPointer([in] int size, [in] int type, [in] int stride, [in, out] void * pointer );
[entry("glTexEnvf")]
void _stdcall glTexEnvf([in] int target, [in] int pname, [in] float param );
[entry("glTexEnvfv")]
void _stdcall glTexEnvfv([in] int target, [in] int pname, [in, out] float * params );
[entry("glTexEnvi")]
void _stdcall glTexEnvi([in] int target, [in] int pname, [in] int param );
[entry("glTexEnviv")]
void _stdcall glTexEnviv([in] int target, [in] int pname, [in, out] int * params );
[entry("glTexGend")]
void _stdcall glTexGend([in] int coord, [in] int pname, [in] double param );
[entry("glTexGendv")]
void _stdcall glTexGendv([in] int coord, [in] int pname, [in, out] double * params );
[entry("glTexGenf")]
void _stdcall glTexGenf([in] int coord, [in] int pname, [in] float param );
[entry("glTexGenfv")]
void _stdcall glTexGenfv([in] int coord, [in] int pname, [in, out] float * params );
[entry("glTexGeni")]
void _stdcall glTexGeni([in] int coord, [in] int pname, [in] int param );
[entry("glTexGeniv")]
void _stdcall glTexGeniv([in] int coord, [in] int pname, [in, out] int * params );
[entry("glTexImage1D")]
void _stdcall glTexImage1D([in] int target, [in] int level, [in] int internalFormat,
   [in] int width, [in] int border, [in] int format, [in] int type, [in, out] void * pixels );
[entry("glTexImage2D")]
void _stdcall glTexImage2D([in] int target, [in] int level, [in] int internalFormat,
 [in] int width, [in] int height, [in] int border, [in] int format, [in] int type, [in, out] void * pixels );
[entry("glTexParameterf")]
void _stdcall glTexParameterf([in] int target, [in] int pname, [in] float param );
[entry("glTexParameterfv")]
void _stdcall glTexParameterfv([in] int target, [in] int pname, [in, out] float * params );
[entry("glTexParameteri")]
void _stdcall glTexParameteri([in] int target, [in] int pname, [in] int param );
[entry("glTexParameteriv")]
void _stdcall glTexParameteriv([in] int target, [in] int pname, [in, out] int * params );
[entry("glTexSubImage1D")]
void _stdcall glTexSubImage1D([in] int target, [in] int level, [in] int xoffset, [in] int width,
 [in] int format, [in] int type, [in, out] void * pixels );
[entry("glTexSubImage2D")]
void _stdcall glTexSubImage2D([in] int target, [in] int level, [in] int xoffset, [in] int yoffset,
 [in] int width, [in] int height, [in] int format, [in] int type, [in, out] void * pixels );
[entry("glTranslated")]
void _stdcall glTranslated([in] double x, [in] double y, [in] double z );
[entry("glTranslatef")]
void _stdcall glTranslatef([in] float x, [in] float y, [in] float z );
[entry("glVertex2d")]
void _stdcall glVertex2d([in] double x, [in] double y );
[entry("glVertex2dv")]
void _stdcall glVertex2dv([in, out] double * v );
```

```
    [entry("glVertex2f")]
    void _stdcall glVertex2f([in] float x, [in] float y );
    [entry("glVertex2fv")]
    void _stdcall glVertex2fv([in, out] float * v );
    [entry("glVertex2i")]
    void _stdcall glVertex2i([in] int x, [in] int y );
    [entry("glVertex2iv")]
    void _stdcall glVertex2iv([in, out] int * v );
    [entry("glVertex2s")]
    void _stdcall glVertex2s([in] short x, [in] short y );
    [entry("glVertex2sv")]
    void _stdcall glVertex2sv([in, out] short * v );
    [entry("glVertex3d")]
    void _stdcall glVertex3d([in] double x, [in] double y, [in] double z );
    [entry("glVertex3dv")]
    void _stdcall glVertex3dv([in, out] double * v );
    [entry("glVertex3f")]
    void _stdcall glVertex3f([in] float x, [in] float y, [in] float z );
    [entry("glVertex3fv")]
    void _stdcall glVertex3fv([in, out] float * v );
    [entry("glVertex3i")]
    void _stdcall glVertex3i([in] int x, [in] int y, [in] int z );
    [entry("glVertex3iv")]
    void _stdcall glVertex3iv([in, out] int * v );
    [entry("glVertex3s")]
    void _stdcall glVertex3s([in] short x, [in] short y, [in] short z );
    [entry("glVertex3sv")]
    void _stdcall glVertex3sv([in, out] short * v );
    [entry("glVertex4d")]
    void _stdcall glVertex4d([in] double x, [in] double y, [in] double z, [in] double w );
    [entry("glVertex4dv")]
    void _stdcall glVertex4dv([in, out] double * v );
    [entry("glVertex4f")]
    void _stdcall glVertex4f([in] float x, [in] float y, [in] float z, [in] float w );
    [entry("glVertex4fv")]
    void _stdcall glVertex4fv([in, out] float * v );
    [entry("glVertex4i")]
    void _stdcall glVertex4i([in] int x, [in] int y, [in] int z, [in] int w );
    [entry("glVertex4iv")]
    void _stdcall glVertex4iv([in, out] int * v );
    [entry("glVertex4s")]
    void _stdcall glVertex4s([in] short x, [in] short y, [in] short z, [in] short w );
    [entry("glVertex4sv")]
    void _stdcall glVertex4sv([in, out] short * v );
    [entry("glVertexPointer")]
    void _stdcall glVertexPointer([in] int size, [in] int type, [in] int stride, [in, out] void * pointer );
    [entry("glViewport")]
    void _stdcall glViewport([in] int x, [in] int y, [in] int width, [in] int height );
  };


  typedef unsigned char * PByte1;


};
```



شکل ۴– Oleview.exe که برای Decompile کردن فایل های tlb بکار می رود .

مثالی از طرز استفاده tlb فوق در VB :

برای استفاده از فایل tlb ایجاد شده ، همانطور که ذکر شد ، در ابتدا باید به کمک Reference در منوی Project ، فایل تولید شده را انتخاب نمود . سپس کد مربوط به فرم برنامه را که در قسمت زیر ارائه شده است ، وارد کنید (توضیحات مربوط به توابع ارائه شده در این قسمت را می توانید در فصول ابتدایی کتاب مطالعه نمایید) :

```
Option Explicit

' +-----------------------------------------------------------------
' | Visual Basic GLCube
' | Ported from the X11 version of glxsimple from SGI's OpenGL site.
' | OpenGL 1.0 type library created by Patrice Scribe.
' | OpenGL is a registered trademark of Silicon Graphics, Inc.
' +-----------------------------------------------------------------

' +-----------------------------------------------------------------
' | First, define all our Win32 BS for getting access to OpenGL and
' | other display whatnots.
' +-----------------------------------------------------------------

Private Type PALETTEENTRY
   peRed As Byte
   peGreen As Byte
   peBlue As Byte
   peFlags As Byte
End Type
Private Type LOGPALETTE
   palVersion As Integer
```

```
    palNumEntries As Integer
    palPalEntry(0 To 255) As PALETTEENTRY
End Type
Private Type PIXELFORMATDESCRIPTOR
    nSize As Integer
    nVersion As Integer
    dwFlags As Long
    iPixelType As Byte
    cColorBits As Byte
    cRedBits As Byte
    cRedShift As Byte
    cGreenBits As Byte
    cGreenShift As Byte
    cBlueBits As Byte
    cBlueShift As Byte
    cAlphaBits As Byte
    cAlphaShift As Byte
    cAccumBits As Byte
    cAccumRedBits As Byte
    cAccumGreenBits As Byte
    cAccumBlueBits As Byte
    cAccumAlpgaBits As Byte
    cDepthBits As Byte
    cStencilBits As Byte
    cAuxBuffers As Byte
    iLayerType As Byte
    bReserved As Byte
    dwLayerMask As Long
    dwVisibleMask As Long
    dwDamageMask As Long
End Type

Const PFD_TYPE_RGBA = 0
Const PFD_TYPE_COLORINDEX = 1
Const PFD_MAIN_PLANE = 0
Const PFD_DOUBLEBUFFER = 1
Const PFD_DRAW_TO_WINDOW = &H4
Const PFD_SUPPORT_OPENGL = &H20
Const PFD_NEED_PALETTE = &H80

Private Declare Function ChoosePixelFormat Lib "gdi32" _
(ByVal hDC As Long, pfd As PIXELFORMATDESCRIPTOR) As Long
Private Declare Function CreatePalette Lib "gdi32" _
(pPal As LOGPALETTE) As Long
Private Declare Sub DeleteObject Lib "gdi32" (hObject As Long)
Private Declare Sub DescribePixelFormat Lib "gdi32" _
(ByVal hDC As Long, ByVal PixelFormat As Long, _
ByVal nBytes As Long, pfd As PIXELFORMATDESCRIPTOR)
Private Declare Function GetDC Lib "gdi32" _
(ByVal hWnd As Long) As Long
Private Declare Function GetPixelFormat Lib "gdi32" _
(ByVal hDC As Long) As Long
Private Declare Sub GetSystemPaletteEntries Lib "gdi32" _
(ByVal hDC As Long, ByVal start As Long, _
ByVal entries As Long, ByVal ptrEntries As Long)
Private Declare Sub RealizePalette Lib "gdi32" _
(ByVal hPalette As Long)
Private Declare Sub SelectPalette Lib "gdi32" _
```

```
(ByVal hDC As Long, ByVal hPalette As Long, ByVal bln As Long)
Private Declare Function SetPixelFormat Lib "gdi32" _
(ByVal hDC As Long, ByVal i As Long, _
pfd As PIXELFORMATDESCRIPTOR) As Boolean
Private Declare Sub SwapBuffers Lib "gdi32" _
(ByVal hDC As Long)
Private Declare Function wglCreateContext Lib "OpenGL32" _
(ByVal hDC As Long) As Long
Private Declare Sub wglDeleteContext Lib "OpenGL32" _
(ByVal hContext As Long)
Private Declare Sub wglMakeCurrent Lib "OpenGL32" _
(ByVal l1 As Long, ByVal l2 As Long)

Dim hPalette As Long
Dim hGLRC As Long

' +-------------------------------------------------------------
' | Set some constants.
' +-------------------------------------------------------------

Dim xAngle As Single
Dim yAngle As Single
Dim zAngle As Single
Dim doubleBuffer As Boolean
Dim displayListInited As Boolean
Sub FatalError(ByVal strMessage As String)


' +-------------------------------------------------------------
' | A simple exit handler should something NASTY happen.
' +-------------------------------------------------------------


   MsgBox "Fatal Error: " & strMessage, vbCritical + vbApplicationModal + vbOKOnly +
vbDefaultButton1, "Fatal Error In " & App.Title
   Unload Me
   Set Form1 = Nothing
   End
End Sub
Sub SetupPixelFormat(ByVal hDC As Long)

' +-------------------------------------------------------------
' | Retrieve/set a Win32 pixel format for OpenGL modes with double-
' | buffering, and direct draw to window with RGBA color mode.
' | 16bit (65536 colors) depth is preferable.
' +-------------------------------------------------------------

   Dim pfd As PIXELFORMATDESCRIPTOR
   Dim PixelFormat As Integer
   pfd.nSize = Len(pfd)
   pfd.nVersion = 1
   pfd.dwFlags = PFD_SUPPORT_OPENGL Or PFD_DRAW_TO_WINDOW Or PFD_DOUBLEBUFFER Or
PFD_TYPE_RGBA
   pfd.iPixelType = PFD_TYPE_RGBA
   pfd.cColorBits = 16
   pfd.cDepthBits = 16
   pfd.iLayerType = PFD_MAIN_PLANE
   PixelFormat = ChoosePixelFormat(hDC, pfd)
```

```
    If PixelFormat = 0 Then FatalError "Could not retrieve pixel format!"
    SetPixelFormat hDC, PixelFormat, pfd
End Sub
Sub SetupPalette(ByVal lhDC As Long)

' +-----------------------------------------------------------------
' | Initialize the Win32 form pallete.
' +-----------------------------------------------------------------

    Dim PixelFormat As Long
    Dim pfd As PIXELFORMATDESCRIPTOR
    Dim pPal As LOGPALETTE
    Dim PaletteSize As Long
    PixelFormat = GetPixelFormat(lhDC)
    DescribePixelFormat lhDC, PixelFormat, Len(pfd), pfd
    If (pfd.dwFlags And PFD_NEED_PALETTE) <> 0 Then
        PaletteSize = 2 ^ pfd.cColorBits
    Else
        Exit Sub
    End If

    pPal.palVersion = &H300
    pPal.palNumEntries = PaletteSize
    Dim redMask As Long
    Dim GreenMask As Long
    Dim BlueMask As Long
    Dim i As Long
    redMask = 2 ^ pfd.cRedBits - 1
    GreenMask = 2 ^ pfd.cGreenBits - 1
    BlueMask = 2 ^ pfd.cBlueBits - 1
    For i = 0 To PaletteSize - 1
        With pPal.palPalEntry(i)
            .peRed = i
            .peGreen = i
            .peBlue = i
            .peFlags = 0
        End With
    Next
    GetSystemPaletteEntries hDC, 0, 256, VarPtr(pPal.palPalEntry(0))
    hPalette = CreatePalette(pPal)
    If hPalette <> 0 Then
        SelectPalette lhDC, hPalette, False
        RealizePalette lhDC
    End If

End Sub
Private Sub Form_KeyPress(KeyAscii As Integer)

' +-----------------------------------------------------------------
' | Handle keypresses.  1 rotates the X axis.  2 rotates the Y axis.
' | 3 handles the Z axis.
' +-----------------------------------------------------------------

    Select Case KeyAscii
        Case Asc("1")
            xAngle = xAngle + 10
            glMatrixMode GL_MODELVIEW
            glLoadIdentity
```

```
            glTranslatef 0, 0, -3
            glRotatef xAngle, 0.1, 0, 0
            glRotatef yAngle, 0, 0.1, 0
            glRotatef zAngle, 0, 0, 1
            Form_Paint
        Case Asc("2")
            yAngle = yAngle + 10
            glMatrixMode GL_MODELVIEW
            glLoadIdentity
            glTranslatef 0, 0, -3
            glRotatef xAngle, 0.1, 0, 0
            glRotatef yAngle, 0, 0.1, 0
            glRotatef zAngle, 0, 0, 1
            Form_Paint
        Case Asc("3")
            zAngle = zAngle + 10
            glMatrixMode GL_MODELVIEW
            glLoadIdentity
            glTranslatef 0, 0, -3
            glRotatef xAngle, 0.1, 0, 0
            glRotatef yAngle, 0, 0.1, 0
            glRotatef zAngle, 0, 0, 1
            Form_Paint
    End Select
End Sub
Private Sub Form_Load()

' +-----------------------------------------------------------------
' | Set angles and initialize.
' +-----------------------------------------------------------------
    Dim hGLRC As Long
    xAngle = 42
    yAngle = 82
    zAngle = 112
    doubleBuffer = GL_TRUE
    displayListInited = GL_FALSE

    SetupPixelFormat hDC
    hGLRC = wglCreateContext(hDC)
    wglMakeCurrent hDC, hGLRC
    glEnable GL_DEPTH_TEST
    glEnable GL_DITHER
    glDepthFunc GL_LESS
    glClearDepth 1
    glClearColor 1, 1, 1, 0
    glMatrixMode GL_PROJECTION
    glLoadIdentity
    glFrustum -1, 1, -1, 1, 1, 10
    glViewport 0, 0, Me.ScaleWidth, Me.ScaleHeight
    displayListInited = GL_FALSE

    glMatrixMode GL_MODELVIEW
    glLoadIdentity
    glTranslatef 0, 0, -3
    glRotatef xAngle, 0.1, 0, 0
    glRotatef yAngle, 0, 0.1, 0
    glRotatef zAngle, 0, 0, 1
    Form_Paint
```

```
End Sub
Private Sub Form_Paint()

' +-----------------------------------------------------------------
' | If a display list has been created, use it.  Otherwise, create it.
' +-----------------------------------------------------------------
   If displayListInited = GL_TRUE Then
      glCallList 1
   Else
      glNewList 1, GL_COMPILE_AND_EXECUTE
      glClear GL_COLOR_BUFFER_BIT Or GL_DEPTH_BUFFER_BIT
      glBegin GL_QUADS

      glColor3f 0, 0.7, 0.1            ' Front Face (Green)
      glVertex3f -1, 1, 1
      glVertex3f 1, 1, 1
      glVertex3f 1, -1, 1
      glVertex3f -1, -1, 1

      glColor3f 0.9, 1, 0              ' Back Face (Yellow)
      glVertex3f -1, 1, -1
      glVertex3f 1, 1, -1
      glVertex3f 1, -1, -1
      glVertex3f -1, -1, -1

      glColor3f 0.2, 0.2, 1            ' Top Side Face (Blue)
      glVertex3f -1, 1, 1
      glVertex3f 1, 1, 1
      glVertex3f 1, 1, -1
      glVertex3f -1, 1, -1

      glColor3f 0.7, 0, 0.1            ' Bottom Side Face (Red)
      glVertex3f -1, -1, 1
      glVertex3f 1, -1, 1
      glVertex3f 1, -1, -1
      glVertex3f -1, -1, -1

      glEnd
      glEndList
      displayListInited = GL_TRUE
   End If

   SwapBuffers hDC
End Sub
Private Sub Form_Resize()
' +-----------------------------------------------------------------
' | Resize the OpenGL view if the form resizes, and redraw.
' +-----------------------------------------------------------------
   glViewport 0, 0, Me.ScaleWidth, Me.ScaleHeight
   Form_Paint
End Sub
Private Sub Form_Unload(Cancel As Integer)
' +-----------------------------------------------------------------
' | Release OpenGL if we decide to quit.
' +-----------------------------------------------------------------
   If hGLRC <> 0 Then
      wglMakeCurrent 0, 0
```

```
      wglDeleteContext hGLRC
   End If

   If hPalette <> 0 Then
      DeleteObject hPalette
   End If

End Sub
Public Sub FillArrayFl(a() As Single, ParamArray n())
Dim v, i&
   For Each v In n
      a(i) = v
      i = i + 1
   Next
End Sub
```

تذکر مهم :

در برنامه های OpenGL تغییر ScaleMode فرم را به VbPixel هیچگاه فراموش نکنید .