Visualib Help Index

Registration Information

License

All versions of Visualib are NOT public Domain software NOR are they free software. Visualib is a copyrighted program and requires the user to register the program if he or she intends to use it except for the purpose of limited evaluation described below.

Registration grants the user a license to use Visualib on a single computer at any one time. A registered user may have Visualib installed on more than one computer, but the program may not be in use on more than one computer at the same time.

No user may modify Visualib in any way, without the written permission of Visual Tech, including, but not limited to, disassembling, debugging or otherwise reverse-engineering the program.

Non-registered users are granted a limited license of 45 days to use Visualib on a trial basis for the purpose of evaluation and determining if Visualib is suitable for their needs. Use of , except for this limited purpose, requires the user to register the product.

All users of Visualib are granted limited license to copy the product only for the trial use by others, subject to the above limitations, provided that Visualib is copied in its full and unmodified form. That is, the copy must include all files necessary to permit full operation of the program, this license agreement, registration form and full documentation. No fee, charge, license, warranty, registration obligation or other compensation of any kind may be accepted by the donor or recipient in exchange for a copy of Visualib.

Operators of Electronic Bulletin Board Systems (BBS Sysops) may permit Visualib to be downloaded by any user, and any user may be permitted to upload a copy of Visualib to a BBS, with the Sysop's permission, provided the above conditions are met.

Use of non-registered copies of Visualib by any person in connection with a business, corporation, educational establishment or government agency is forbidden. Such users must register the product.

Warranty

Visual Tech makes no warranty of any kind, express or implied, as to the suitability of the product for a particular purpose and shall not be liable for any damages, loss of productivity, loss of profits or savings or any other incidental or consequential damages, whether direct, indirect or consequential, arising from any failure of the product to operate in any manner desired by the user for which it was not intended or as a result of the user's inability or failure to use the program in the manner in which it was intended. Visual Tech shall not be liable for any damage to data or property which may be caused directly or indirectly by use of the program.

Order Form

Visual Tech Co.

P.O. Box 8735

Fort Wayne, IN 46898-8735

(219) 489-0235

Product	Quantity	Unit Price	Amount
Visualib 2.0 for Windows (DLL)	_____	$399.00	$ _____
Visualib 1.x for Windows (Microsoft C version)	_____	$50.00	$ _____
Visualib 1.x for Windows (Borland C++ version)	_____	$50.00	$ _____
Visualib 1.x for DOS (Microsoft C version)	_____	$40.00	$ _____
Visualib 1.x for DOS (Borland C++ version)	_____	$40.00	$ _____
		Subtotal	$ _____
		Tax	$ _____
(Indiana Residents must add 5.0 % sales tax)			
		Shipping	$ 5.00
		TOTAL	$ _____

Disk Format : ( )5.25" ( )3.5"

Name : _____

Company : _____

Address : _____

_____

City : _____ State : _____ Zip : _____

Overview

Visualib 2.0 is a comprehensive state-of-the-art graphics library for the Microsoft Windows environment. It contains powerful and efficient functions for rendering both 2D and 3D graphic objects. Visualib 2.0 consists of several DLLs which can be used with any Microsoft Windows develop environments such as Microsoft C/C++ , Microsoft Visual Basic, and Borland C++ version 2.0 and up.

Complete 2D and 3D viewing systems allows flexible view settings. Sophisticated transformation mechanism supports virtually all types of graphics transformations. An object transformation stack is maintained in conjunction with the transformation functions to achieve flexible and efficient graphic effects.

Visualib provides many different lighting, shading, material, and other rendering options. Lights can be created individually with different characteristics. Various shading options including Gouraud shading and Phong shading are supported. Materials of different characteristics can also be created and selected for different objects. Double buffering is supported for both 2D and 3D viewers. z-buffer is also available to handle complex backface eliminations.

Visualib supports a full set of common 2D and 3D drawing functions and the powerful curve and surface drawing functions such as Bezier, Hermit curves, B-Spline, NURBS curves and surfaces. Visualib also includes a large collection of graphics primitives.

Image mapping is available to map standard Windows bitmaps to graphics objects. Visualib also provides texture mapping functions to render 3D solid textures.

Visualib contains a set of functions to display true 3D text using any TrueType font. All the shading modes are available in text display. Object transformations can also be applied to affect the character display.

Visualib Programming Guide

Getting Started

Visualib 2.0 disk contains the following files:

README.1ST - read me first
REGISTER.TXT - ASCII registration form
VISUALIB.LIB - import library file
VISUALIB.DLL - dynamic link library file
VISUALIB.H - header file
VISUALIB.HLP - on-line Windows help of Visualib
VLIBDEMO.C - Visualib demo program source code
VLIBDEMO.RC - Visualib demo program resource file
VLIBDEMO.DEF - Visualib demo program module definition file
VLIBDEMO.EXE - Visualib demo program executable

The best place to start your Visualib programming is the demo program VLIBDEMO included in the distribution disk. The executable file is ready to run in Windows. Try it and enjoy the show!

The source code VLIBDEMO.C illustrates the application of Visualib library to create beautiful graphics applications. It uses many features on Visualib and may serve as a template of Visualib applications.

Visualib functions are contained in the library file VISUALIB.LIB. Place it in a directory so that your linker can find it. In order to use the library functions in your Windows program, the header file VISUALIB.H needs to be included in your C source code after WINDOWS.H.

To use the Visualib system, first you need to initialize the graphics system by calling InitializeVisualib. After the graphics system is initialized, you may create 2D or 3D viewers by calling CreateViewer. Then call the viewing transformation functions and projection transformation functions to setup the viewers.

Now you can start to draw graphics through the viewers. Using the rich set of drawing functions provided by Visualib together with the modeling transformation functions and the matrix stacks, you will be able to achieve most sophisticated visual effects with ease.

Call the function ExitVisualib to exit the Visualib system.

Visualib Initialization and Termination

The following initialization function should be called before using the Visualib systems.

InitializeVisualib

The initialization function allocates and initializes necessary system variables.

To exit a Visualib graphics system, use the function

ExitVisualib

ExitVisualib frees all the memory used by the Visualib system.

Coordinate Systems

Visualib has several different coordinate systems that concern users.

The world coordinate system is the common coordinate system referenced by all parts of Visualib. It is a logical 2D or 3D coordinate system which many Visualib functions specify the viewers and geometric objects. You may define the world coordinates in any way to suit your application. It does not need to be correlated to the display configuration. Because of the powerful viewing transformations of Visualib, you can set up arbitrary viewing configurations in any world coordinates. The axes of a 3D world coordinate system may be displayed by calling the function:

MarkPosition3D

A local coordinate system (or object coordinate system) is a system attached to a set of objects. The world coordinates of the objects are obtained through object transformation.

The screen coordinate system is the coordinate system used in MS Windows GDI functions. Several Visualib functions use this system to specify certain parameters related to the display devices. Because Visualib is compatible with the GDI functions, user may also call some GDI functions with this kind of coordinates while using Visualib.

The viewing coordinate system is an intermediate coordinate system used by Visualib. The following viewing transformations may be best thought of as operations in the viewing coordinate system.

MoveViewer3D
RotateViewer3D
ZoomViewer3D
MoveViewer2D
RotateViewer2D

There are two types of coordinates used to specify points in the world coordinate space: The Euclidean coordinates and the homogeneous coordinates.

Three floating point numbers (x,y,w) are used to define a 2D point and four floating point numbers (x,y,z,w) are used for a 3D point. A point in the 2D space with homogeneous coordinate (x,y,w) corresponds to the Euclidean coordinate (x/w,y/w) and a 3D point with homogeneous coordinate (x,y,z,w) corresponds the Euclidean coordinate (x/w, y/w, z/w). Although the homogeneous representation will take a little more memory. There are many advantages associated with the homogeneous coordinates:

All affine transformations (including translation) can be handled in a uniform manner by linear transformations.

Perspective projections can be applied naturally and with the clipping in the homogeneous coordinates, the overflow problem associated with the perspective projections is avoided.

For the NURBS curves and surfaces, it is necessary to specify the homogenous coordinates.

Viewer

A viewer is a logical structure which specifies precisely how the graphics objects in a world coordinate system (2D or 3D) is displayed in a two dimensional screen viewport.

Viewport

The viewport of a viewer is a rectangular region in a window client area which is used for the actual display of the content of the viewer.

Viewer Position

The viewer position defines the position and view direction of the viewer in the world coordinate system.

Projection

Projection defines the view volume and the way it is mapped to the viewport. A 3D projection is either perspective or orthogonal. It also specifies the depth clipping region.

Viewer Setup

User can establish virtually unlimited number of independent 2D and 3D viewers. In each viewer, user can select various parameters such as viewport, viewer position and directions, perspective or orthogonal projections, depth of view volume, etc.

A 2D or 3D viewer contains three major components:

Viewport
Viewer Position
Projection

The following are viewer setup functions.

CreateViewer
SetViewport
SetViewerName
DisplayViewerFrame
DisplayViewerName
SetView2D
SetProjection2D
SetWindow
SetView3D
SetPolarView
SetPerspective
SelectViewer
ClearViewer

The viewing transformations may be modified by the following functions

MoveViewer2D
RotateViewer2D
ZoomViewer2D
MoveViewer3D
RotateViewer3D
ZoomViewer3D

Note that the viewing transformations are different from the modeling transformations. The modeling transformations affect the current transformation matrix on the stack top only, while the viewing transformations change the setting of a viewer.

To get information on a viewer, use the following functions:

NumViewer
ViewerLocation
ViewerDirection
ViewerField3D

Modeling Transformations and Matrix Stack

Transformations are an important part of the graphics system.Visualib provides a sophisticated transformation mechanism to support virtually all types of graphics transformations. Users may arbitrarily translate, scale, or rotate any object in any sequence. Visualib maintains a transformation stack which can be used in conjunction with the transformation functions to achieve flexible and efficient graphic effects.

Rotate2D
PointRotate2D
Translate2D
TranslateTo2D
Scale2D
PointScale2D
Shear2D
Stretch2D
Mirror2D

Rotate3D
AxleRotate3D
Translate3D
TranslateTo3D
Scale3D
PointScale3D
Shear3D
Stretch3D
Mirror3D

Note that the modeling transformations are different from the viewing transformations. The modeling transformations affect the current transformation matrix on the stack top only, while the viewing transformations change the setting of a viewer.

To systematically manage the transformation processes, Visualib provides transformation stacks for 2D and 3D modeling transformations. The stack top determines the final effect of transformation process. All the transformation functions discussed above changes some aspects of the stack top. To save the current transformation configurations, use the following functions

PushTransforamtion2D
PushTransformation3D

These functions will push the current stack top and leave the stack top unchanged. You may get back to this particular state later by using the following function.

PopTransformation2D
PopTransformation3D

Lighting Models

Visualib contains an advanced lighting and shading system for rendering graphics objects. Visualib provides many different lighting, shading, material, and other rendering options. With various combinations of the options, dramatic visual effects can be achieved. Users may create virtually unlimited number of lights and individually specify the characteristics such as position, direction, colors, intensity, global or local lights. Graphics objects may be rendered in many different ways. Various shading modes such as flat shading, solid fill, Gouraud shading, and Phong shading are supported. Materials of different characteristics can also be created and selected for different objects.

Unlimited number of lights of various characteristics can be created. Position, direction, color, intensity, and other properties can be individually set. Each light can be turned on or off at any time.

CreateLight
DeleteLight
CopyLight
SelectLight
SwitchLight

Users may select several different shading options, from simple wire-frame and flat shading to complicated Gouraud shading and Phong shading.

CreateLModel
DeleteLModel
CopyLModel
SelectLModel

In a similar way, different materials can be created and selected for different objects.

CreateMaterial
DeleteMaterial
CopyMaterial
SelectMaterial

The following functions set or get various shading options and parameters.

ShadingOption
ShadingParameter
ShadingColor
ShadingFactor

Double Buffer and z-Buffer

Double buffering is supported for both 2D and 3D viewers. User may select double buffer mode to achieve smooth animation effects. Advanced hidden surface elimination techniques are employed in Visualib. Backface culling may be used for simple polygonal surfaces and z-buffer may selected to handle arbitrarily complex views.

The following functions provide double buffer support.

BeginDoubleBuffer
EndDoubleBuffer
UpdateDoubleBuffer

Depth buffer, or z-buffer, is a general technique to achieve hidden surface removal. Visualib provides the following functions to support z-buffer.

SetDepthBuffer
ClearDepthBuffer

Drawing Functions

Visualib supports a full set of common 2D and 3D drawing functions such as lines, polygons, ellipses, spheres, polyhedra, etc.

Visualib greatly extends the capabilities of windows' GDI functions. For example, Visualib uses floating point type for specifying coordinates and implements clipping in homogeneous coordinates, which effectively avoids the common integer overflow problem associated with the perspective viewing. However, all GDI functions are still available and the function calls from both systems can be used at the same time. Visualib can be used with any types of device context - screens, printers, or memory. Consequently, the same routine for display can also be used for printing or storing. Visualib also uses the attributes such as colors, line width of the device context set by the GDI functions.

MoveTo2D
LineTo2D
Line2D
RMoveTo2D
RLineTo2D
RLine2D
MoveTo2H
LineTo2H
Polyline2D
ClosedPolyline2D
Polygon2D
Rectangle2D
Disk2D
Arc2D
Wedge2D
Ngon2D
Star2D
Flower2D

MoveTo3D
LineTo3D
Line3D
RMoveTo3D
RLineTo3D
RLine3D
MoveTo3H
LineTo3H
Polyline3D
ClosedPolyline3D
Polygon3D
Rectangle3D

Curves and Surfaces

Curves and surfaces have become important parts of advanced graphics systems. They offer powerful and flexible ways to specify complicated objects with various smoothness and continuity. Visualib provides the powerful curve and surface drawing functions such as cubic Bezier, Hermit, B-Spline, and NURBS curves and surfaces.

BezierCurve2D
HermitCurve2D
BSplineCurve2D
NURBSCurve2D
BSplineCurveClosed2D
NURBSCurveClosed2D
CatmullRomSpline2D
BezierCurve3D
HermitCurve3D
BSplineCurve3D
NURBSCurve3D
BezierSurface
HermitSurface
BSplineSurface
NURBSSurface
CoonsPatch

Graphics Primitives

Visualib includes a large collection of 2D and 3D graphics primitives. Many graphic objects can be drawn with a simple function call.

Cube
Cylinder
Cone
Ellipsoid
Sphere
HemiSphere
SolidStar
SolidFlower
Wedge
Frustum
Ridge
Prism
Pyramid
Tetrahedron
Dodecahedron
Icosahedron
Octahedron

Parabola
Hyperbola
OscillatoryWave
Catenary
Spiral2D
Cycloid
Epicycloid
Cardioid
Hypocycloid
Lemniscate
Rose

Spring
Spiral3D
EllipticParaboloid
Hyperboloid1
Hyperboloid2
HyperbolicParaboloid

Image and Texture Mapping

Image mapping is a useful technique to significantly enhance the visual effects. Visualib allows user to map standard Windows bitmaps to graphics objects. The images will be transformed appropriately to achieve the correct perspective view. Visualib also provides texture mapping functions to render 3D solid textures.

Visualib provides the following functions for mapping a device independent bitmap to a 2D or 3D object.

ImageMap2D
ImageMap3D

3D texture mapping is another powerful feature of Visualib.

SolidTexture

2D and 3D Text

Visualib contains a set of functions to display true 3D solid texts as well as 2D and 3D flat texts using any TrueType font. All the shading modes are available in solid text display. Object transformations can also be applied to affect the character display.

SetFont
TextParameter
DrawString

Coordinate Type

Visualib defines four different coordinate types for points.

VL_2D
VL_2H
VL_3D
VL_3H

VL_2D uses two floating numbers to specify a 2D Euclidean point.
VL_2H uses three floating numbers to specify a 2D homogeneous point.
VL_3D uses three floating numbers to specify a 3D Euclidean point.
VL_3H uses four floating numbers to specify a 3D homogeneous point.

Visualib Function Reference

E

F

G

H

I

L

M

Tetrahedron

Function

Draws a tetrahedron.

Syntax

void Tetrahedron (HDC hdc, float r);

Remarks

Tetrahedron draws a tetrahedron in the current 3D viewer with current pen color for the edges and current brush color for the interior. *r* specifies the radius of the circumscribing sphere.

Return Value

None.

See also

Octahedron, Dodecahedron, Icosahedron

Octahedron

Function

Draws an octahedron.

Syntax

void Octahedron (HDC hdc, float r);

Remarks

Octahedron draws an octahedron in the current 3D viewer with current pen color for the edges and current brush color for the interior. *r* specifies the radius of the circumscribing sphere.

Return Value

None.

See also

Tetrahedron, Dodecahedron, Icosahedron

Dodecahedron

Function

Draws a dodecahedron.

Syntax

void Dodecahedron (HDC hdc, float r);

Remarks

Dodecahedron draws a dodecahedron in the current 3D viewer with current pen color for the edges and current brush color for the interior. *r* specifies the radius of the circumscribing sphere.

Return Value

None.

See also

Tetrahedron, Octahedron, Icosahedron

Icosahedron

Function

Draws an icosahedron.

Syntax

void Icosahedron (HDC hdc, float r);

Remarks

Icosahedron draws an icosahedron in the current 3D viewer with current pen color for the edges and current brush color for the interior. *r* specifies the radius of the circumscribing sphere.

Return Value

None.

See also

Tetrahedron, Octahedron, Dodecahedron

InitializeVisualib

Function

Initializes the graphic system.

Syntax

BOOL InitializeVisualib (void);

Remarks

InitializeVisualib initializes Visualib graphic system. It must be called before any other Visualib functions.

Return value

On successful completion, InitalizeVisualib returns TRUE. It returns FALSE on error.

See also

ExitVisualb

ExitVisualib

Function

Exits the graphic system and frees memory.

Syntax

void ExitVisualib (void);

Remarks

ExitVisualib exits the graphics systems. The memory allocated by Visualib is released.

Return value

None.

See Also

InitializeVisualib

PenColor

Function

Selects a pen color.

Syntax

HPEN PenColor (HDC hdc, short color);

Remarks

PenColor selects a system pen with color index for the current device context.

Return value

PenColor returns a handle to the previously selected pen.

See also

BrushColor

BrushColor

Function

Selects a brush color.

Syntax

HBRUSH BrushColor (HDC hdc, short color);

Remarks

BrushColor selects a system brush with color index for the current device context.

Return value

BrushColor returns a handle to the previously selected brush.

See also

PenColor

TextColor

Function

Sets text color.

Syntax

void TextColor (HDC hdc, int tcolor, int bcolor, int mode);

Remarks

TextColor sets the text color, the background color, and the background mode to *tcolor, bcolor*, and *mode*.

Return value

None.

See also

Label2D, Label3D

CreateViewer

Function

Creates a 2D or 3D viewer

Syntax

int CreateViewer (NPSTR name, RECT port, BYTE type, BYTE mode);

Remarks

CreateViewer creates a viewer. The viewport is defined by *port* . The name of the viewer is given by string *name*. *type* defines 2D or 3D viewer which takes one of the following values.
VL_TWOD
VL_THREED

mode defines the way to fit the viewport which takes following values.

VL_HORIZONTALFIT	fit the horizontal size and keep aspect ratio
VL_VERTICALFIT	fit the vertical size and keep aspect ratio
VL_AUTOFIT	fit automatically to include entire view and keep aspect ratio
VL_VIEWPORTFIT	stretch the view to fit the viewport

Return Value

The viewer handle will be returned if it is created successfully. Otherwise, NULL will be returned. The viewer handle is used for other Visualib functions to reference the viewer.

See also

InitializeVisualib, SetViewport

DeleteViewer

Function

Deletes a viewer.

Syntax

void DeleteViewer (int Viewer)

Remarks

DeleteViewer deletes the viewer specified.

Return value

None

See also

CreateViewer

CopyViewer

Function

Copy viewers.

Syntax

BOOL CopyViewer (int viewdst, int viewsrc)

Remarks

CopyViewer copies the content of *viewsrc* to *viewdst*.

Return value

CopyViewer returns TRUE if successful and FALSE on error.

See also

CreateViewer

SelectViewer

Function

Selects a viewer.

Syntax

BOOL SelectViewer (int hview);

Remarks

SelectViewer selects viewer hview as the current viewer. The subsequent drawing function calls will use this viewer. *hview* must be a valid viewer handle returned by CreateViewer.

Return value

On success, SelectViewer returns TRUE. On error, it returns FALSE.

See also

CreateViewer

DisplayViewerFrame

Function

Displays the frame of a viewer.

Syntax

BOOL DisplayViewerFrame (HDC hdc, int hview);

Remarks

DisplayViewerFrame draws the viewer hview's rectangular border with current pen color. The frame is defined by the viewport set in the function CreateViewer or SetViewport.

Return value

On success, DisplayViewerFrame returns TRUE. On error, it returns FALSE.

See also

CreateViewer, SetViewport

DisplayViewerName

Function

Display viewer's name.

Syntax

BOOL DisplayViewerName (HDC hdc, int hview, int top);

Remarks

DisplayViewerName displays the viewer *hview*'s name string. The name is displayed on the top of the viewport if the parameter *top* is nonzero.

Return value

On success, DisplayViewerName returns TRUE. On error, it returns FALSE.

See also

GetViewerName, SetViewerName

Label2D

Function

Draws a label.

Syntax

void Label2D (HDC hdc, float x, float y, LPSTR label);

Remarks

Label2D draws a label in the current 2D viewer. The starting point is defined by (*x, y*).

Return value

None.

See also

Label3D

Label3D

Function

Draws a label.

Syntax

void Label3D (HDC hdc, float x, float y, float z, LPSTR label);

Remarks

Label3D draws a label in the current 3D viewer starting at (*x, y, z*).

Return value

None.

See also

Label2D

ClearViewer

Function

Clears a viewer.

Syntax

BOOL ClearViewer (HDC hdc, int hview, int color);

Remarks

ClearViewer clears the viewer *hview*'s client area with *color*.

Return value

On success, ClearViewer returns TRUE. On error, it returns FALSE.

See also

CreateViewer

NumViewer

Function

Gets the number of viewers.

Syntax

short NumViewer (void);

Remarks

NumViewer returns the number of viewers currently created.

Return value

The number of viewers.

See also

CreateViewer

ViewerMappingMode

Function

Sets viewer mapping mode.

Syntax

int ViewerMappingMode (int viewer, int mode)

Remarks

ViewerMappingMode sets the viewer mapping mode for the specified *viewer*. The previous mapping mode is returned.

Return value

The previous mapping mode.

See also

CreateViewer

PushTransformation2D

Function

Pushes the 2D transformation matrix stack.

Syntax

BOOL PushTransformation2D (MATRIX m);

Remarks

PushTransformation2D pushes the 2D object transformation matrix stack. The stack top is the product of the previous stack top and *m*. If *m* is NULL a copy of the previous stack top is pushed to the stack.

Return value

PushTransformation2D returns TRUE upon successful completion. FALSE is returned if the stack is full.

See also

PopTransformation2D

PopTransformation2D

Function

Pops the 2D transformation matrix stack.

Syntax

BOOL PopTransformation2D (MATRIX m);

Remarks

PopTransformation2D pops the 2D object transformation matrix stack. The stack top is assigned to *m*. If *m* is NULL the stack top is discarded.

Return value

On success, PopTransformation2D returns TRUE. FALSE is returned if the stack is empty.

See also

PushTransformation2D

LoadTransformation2D

Function

Loads a 2D transformation matrix.

Syntax

void LoadTransformation2D (MATRIX m);

Remarks

LoadTransformation2D replaces the 2D transformation matrix stack top with matrix *m*.

Return value

None.

See also

PushTransformation2D

PushTransformation3D

Function

Pushes the 3D transformation matrix stack.

Syntax

BOOL PushTransformation3D (MATRIX m);

Remarks

PushTransformation3D pushes the 3D object transformation matrix stack. The new stack top is the product of the previous stack top and the matrix *m*. If *m* is NULL a copy of previous stack top is pushed to the stack.

Return value

On success, PopMatrix3D returns TRUE. FALSE is returned if the stack is full.

See also

PopMatrix3D

PopTransformation3D

Function

Pops the 3D transformation matrix stack.

Syntax

BOOL PopTransformation3D (MATRIX m);

Remarks

PopTransformation3D pops the 3D object transformation matrix stack. The stack top is assigned to *m*. If *m* is NULL the stack top is discarded.

Return value

On success, PopTransformation3D returns TRUE. FALSE is returned if the stack is empty.

See also

PushTransformation3D

LoadTransformation3D

Function

Loads a 3D transformation matrix.

Syntax

void LoadTransformation3D (MATRIX m);

Remarks

LoadTransformation3D replaces the 3D object transformation stack top by the matrix *m*. The current stack top is discarded.

Return value

None.

See also

PushTransformation3D

SetView3D

Function

Sets a 3D viewer's view transformation.

Syntax

BOOL SetView3D (int hview, float vx, float vy, float vz, float rx, float ry, float rz, float twist);

Remarks

SetView3D sets the 3D viewer *hview*'s viewing transformation matrix according to the viewer position (*vx, vy, vz*), a view reference point (*rx, ry, rz*), and the *twist* angle.

Return value

On success, SetView3D returns TRUE. On error, it returns FALSE.

See also

SetPolarView

SetPolarView

Function

Sets 3D view transformation based on polar coordinates.

Syntax

BOOL SetPolarView (int hview, float cx, float cy, float cz, float dist, float azim, float inc, float twist);

Remarks

SetPolarView sets the 3D viewer *hview*'s view transformation according to the reference center (cx, cy, cz), the distance *dist* from the reference center to the eye postition, and the three orientation angles *azim*, *inc*, and *twist*.

Return value

On success, SetPolarView returns TRUE. On error, it returns FALSE.

See also

SetView3D

SetPerspective

Function

Sets perspective projection of a 3D viewer.

Syntax

BOOL SetPerspective (int hview, float fovy, float aspect, float front, float back);

Remarks

SetPersperspective sets 3D viewer *hview*'s perspective projection matrix according to the field of view angle *fovy*, aspect ratio *aspect*, *front* and *back* clipping panes.

Return value

On success, SetPerspective returns TRUE. On error, it returns FALSE.

See also

SetProjection3D

SetProjection3D

Function

Sets projection of a 3D viewer.

Syntax

BOOL SetProjectin3D (int hview, float left, float right, float bottom, float top, float front, float back, BYTE mode);

Remarks

SetProjection3D sets 3D viewer *hview*'s projection according to the viewing box defined by the prarameters *left*, *right*, *bottom*, *top*, *front*, and *back*. *mode* defines the projection mode which is one of the following values.
VL_PERSPECTIVE
VL_ORTHOGONAL

Return value

On success, SetProjection3D returns TRUE. On error, it returns FALSE.

See also

SetPerspective

SetViewport

Function

Sets a viewer's viewport.

Syntax

BOOL SetViewport (int hview, RECT port);

Remarks

SetViewport sets viewer *hview*'s viewport to the rectangle *port* in display coordinates.

Return value

On success, SetViewport returns TRUE. On error, it returns FALSE.

See also

GetViewport

SetView2D

Function

Sets a 2D viewer's view transformation.

Syntax

BOOL SetView2D (int hview, float x, float y, float angle);

Remarks

SetView2D sets 2D viewer *hview*'s view transformation according to the center coordinates (*x,y*), and the rotation *angle*.

Return value

On success, SetView2D returns TRUE. On error, it returns FALSE.

See also

SetProjecton2D

SetProjection2D

Function

Sets 2D viewer's projection transformation.

Syntax

BOOL SetProjection2D (int hview, float left, float right, float bottom, float top);

Remarks

SetProjection2D sets 2D viewer *hview*'s projection transformation according to the two corner points of the projection rectangle defined by *left*, *right*, *bottom*, and *top*.

Return value

On success, SetProjection2D returns TRUE. On error, it returns FALSE.

See also

SetView2D

SetWindow

Function

Sets 2D viewer's view and projection transformations.

Syntax

BOOL SetWindow (int hview, float x1, float y1, float x2, float y2);

Remarks

SetWindow sets 2D viewer *hview*'s view transformation and projection transformation according to the two corner points in the world coordinates (*x1, y1*) and (*x2, y2*)..

Return value

On success, SetWindow returns TRUE. On error, it returns FALSE.

See also

SetView2D, SetProjection2D

MoveViewer3D

Function

Moves a 3D viewer.

Syntax

BOOL MoveViewer3D (int hview, float dx, float dy, float dz, BOOL viewcoord);

Remarks

MoveViewer3D moves the 3D viewer *hview* by the amount *dx, dy, dz* in the view coordinate system. If *viewcoord* is TRUE, the move is about the view coordinate system. If *viewcoord* is FALSE, the move is about the world coordinate system.

Return value

On success, MoveViewer3D returns TRUE. On error, it returns FALSE.

See also

RotateViewer3D, ZoomViewer3D

RotateViewer3D

Function

Rotates a 3D viewer.

Syntax

BOOL RotateViewer3D (int hview, float yaw, float pitch, float twist, BOOL viewcoord);

Remarks

RotateViewer3D rotates the 3D viewer *hview* in the view coordinate system according to angles of yaw, pitch, and twist in degrees. If *viewcoord* is TRUE, the rotation is about the view coordinate system. If *viewcoord* is FALSE, the rotation is about the world coordinate system.

Return value

On success, RotateViewer3D returns TRUE. On error, it returns FALSE.

See also

MoveViewer3D, ZoomViewer3D

ZoomViewer3D

Function

Zooms a 3D viewer.

Syntax

BOOL ZoomViewer3D (int hview, float zoom);

Remarks

ZoomViewer3D zooms the 3D viewer hview by the factor *zoom*.

Return value

On success, ZoomViewer3D returns TRUE. On error, it returns FALSE.

See also

MoveViewer3D, RotateViewer3D

MoveViewer2D

Function

Moves a 2D viewer.

Syntax

BOOL MoveViewer2D (int hview, float dx, float dy, BOOL viewcoord);

Remarks

MoveViewer2D moves a 2D viewer *hview* by the amount *dx* and *dy*. If *viewcoord* is TRUE, the move is about the view coordinate system. If *viewcoord* is FALSE, the move is about the world coordinate system.

Return value

On success, MoveViewer2D returns TRUE. On error, it returns FALSE.

See also

RotateViewer2D, ZoomViewer2D

RotateViewer2D

Function

Rotates a 2D viewer.

Syntax

BOOL RotateViewer2D (int hview, float angle, BOOL viewcoord);

Remarks

RotateViewer2D rotates the 2D viewer *hview* by *angle* in degrees in the view coordinate system. If *viewcoord* is TRUE, the rotation is about the view coordinate system. If *viewcoord* is FALSE, the rotation is about the world coordinate system.

Return value

On success, RotateViewer2D returns TRUE. On error, it returns FALSE.

See also

MoveViewer2D, ZoomViewer2D

ZoomViewer2D

Function

Zooms a 2D viewer.

Syntax

BOOL ZoomViewer2D (int hview, float zoom);

Remarks

ZoomViewer2D zooms the 2D viewer *hview* by the factor *zoom*.

Return value

On success, ZoomViewer2D returns TRUE. On error, it returns FALSE.

See also

MoveViewer2D, RotateViewer2D

ViewerLocation

Function

Gets a 3D viewer's position.

Syntax

BOOL ViewerLocation (int hview, float FAR *vx, float FAR *vy, float FAR *vz);

Remarks

ViewerLocation gets the 3D viewer *hview*'s position in the world coordinate system.

Return value

On success, ViewerLocation returns TRUE. On error, it returns FALSE.

See also

SetView3D

ViewerDirection

Function

Gets a 3D viewer's direction.

Syntax

BOOL ViewerDirection (int hview, float FAR *x, float FAR *y, float FAR *z);

Remarks

ViewerDirection gets the 3D viewer *hview*'s view direction in the world coordinate system .

Return value

On success, ViewerDirection returns TRUE. On error, it returns FALSE.

See also

SetView3D, SetPolarView

ViewerField3D

Function

Gets a 3D viewer's view field.

Syntax

BOOL ViewerField3D (int hview, float *left, float *right, float *bottom, float *top, float *front, float *back);

Remarks

ViewerField3D gets the 3D viewer *hview*'s view field defined by *left*, *right*, *bottom*, *top*, *front*, and *back* in the view coordinate system.

Return value

On success, ViewerField3D returns TRUE. On error, it returns FALSE.

See also

SetPespective, SetProjection3D

ViewerField2D

Function

Gets a 2D viewer's viewing field.

Syntax

short ViewerField2D (int hview, float *left, float *right, float *bottom, float *top);

Remarks

ViewerField2D gets the 2D viewer hview's viewing field defined by left, right, bottom, and top in the view coordinate system.

Return value

On success, ViewerField2D returns 0. On error, it returns a nonzero value.

See also

SetProjection2D

Rotate3D

Function

Rotates on the current transformation matrix.

Syntax

void Rotate3D (float angle, char axis);

Remarks

Rotate3D performs a 3D object rotation about *axis*. It changes the current 3D transformation matrix (the stack top). *axis* can be 'x;, 'y', or 'z'. *angle* is measured in degrees.

Return value

None

See also

Translate3D, Scale3D

AxleRotate3D

Function

Rotates about an arbitrary axis.

Syntax

void AxleRotate (float angle, VECTOR point, VECTOR direction);

Remarks

AxleRotate3D performs a 3D object transformation of rotating by *angle* about the axis defined by *point* and *direction*.

Return value

None.

See also

Rotate3D

Translate3D

Function

Translates on the current 3D transformation matrix.

Syntax

void Translate3D (float x, float y, float z);

Remarks

Translate3D performs a 3D object transformation on the current 3D transformation matrix by a translation of amount (*x, y, z*).

Return value

None.

See also

Rotate3D, Scale3D

TranslateTo3D

Function

Translates to a point.

Syntax

void TranslateTo3D (float x, float y, float z);

Remarks

TranslateTo3D performs the object transformation of translating the origin to (*x, y , z*).

Return value

None.

See also

Translate3D

Scale3D

Function

Scales on the current 3D transformation matrix.

Syntax

void Scale3D (float sx, float sy, float sz);

Remarks

Scale3D scales on the current 3D transformation matrix (the stack top) in the x , y, and z directions by the amount *sx, sy*, and *sz*.

Return value

None.

See also

Translate3D, Rotate3D

PointScale3D

Function

Scales about a point.

Syntax

void PointScale3D (float sx, float sy, float sz, VECTOR point);

Remarks

PointScale3D performs the object transformation of scaling by *sx, sy, sz* about *point*.

Return value

None.

See also

Scale3D

Stretch3D

Function

Stretchs along a line.

Syntax

void Stretch3D (float factor, VECTOR point, VECTOR direction);

Remarks

Stretch3D performs a 3D objection transformation of stretching by the amount *factor* about the plane defined by *point* and *direction*.

Return value

None.

See also

Shear3D

Shear3D

Function

Performs a 3D shear operation.

Syntax

void Shear3D (float factor, VECTOR point, VECTOR normal, VECTOR direction);

Remarks

Shear3D performs the object transformation of shearing by the amount *factor* about the plane defined by *point* and *normal* along *direction*.

Return value

None.

See also

Stretch3D

Mirror3D

Function

Performs a mirror reflection.

Syntax

void Mirror3D (VECTOR point, VECTOR normal);

Remarks

Mirror3D performs a 3D object transformation of mirror reflection about the plane defined by *point* and *normal*.

Return value

None.

See also

Shear3D

Translate2D

Function

Translates on the current 2D transformation matrix.

Syntax

void Translate2D (float x, float y);

Remarks

Translate2D performs a 2D objec translation of the amount (*x, y*).

Return value

None.

See also

Rotate2D, Scale2D

TranslateTo2D

Function

Translates the origin.

Syntax

void TranslateTo2D (float x, float y);

Remarks

TranslateTo2D translates the origin to (*x, y*) in the world coordinate system.

Return value

None.

See also

Translate2D

Rotate2D

Function

Rotates on the current 2D transformation matrix.

Syntax

void Rotate2D (float angle);

Remarks

Rotate2D rotates on the current 2D transformation matrix (the stack top) by the amount *angle*.

Return value

None.

See also

Translate2D, Scale2D

PointRotate2D

Function

Rotates about a point.

Syntax

void PointRotate (float angle, float x, float y);

Remarks

PointRotate2D performs a rotation about the point (*x, y*) of the amount *angle*.

Return value

None.

See also

Rotate2D

Scale2D

Function

Scales on the current 2D transformation matrix.

Syntax

void Scale2D (float sx, float sy);

Remarks

Scale2D scales on the current 2D transformation matrix (the stack top) in the x and y directions by the amount (*sx, sy*).

Return value

None.

See also

Translate2D, Rotate2D

PointScale2D

Function

Scales about a point.

Syntax

void PointScale2D (float x, float y, float sx, float sy);

Remarks

PointScale2D performs a 2D scaling about the point (*x, y*) of factors *sx* and *sy* in x and y directions respectively.

Return value

None.

See also

Scale2D

Shear2D

Function

Performs a 2D shear transformation.

Syntax

void Shear2D (float factor, float x, float y, float angle);

Remarks

Shear2D performs a 2D object shear transformation. (*x,y*) is the center of transformation. The axis perpendicular to the direction of shearing is defined by *angle*. The amount of shearing is given by *factor*.

Return value

None.

See also

Translate2D, Scale2D, Rotate2D

Stretch2D

Function

Performs a stretch object transformation.

Syntax

void Stretch2D (float factor, float x, float y, float angle);

Remarks

Stretch2D performs a 2D stretch object transformation about the point (*x, y*) and along the line defined by *angle*.

Return value

None.

See also

Shear2D

Mirror2D

Function

Performs a mirror reflection.

Syntax

void Mirror2D (float x, float y, float angle);

Remarks

Mirror2D performs a mirror reflection about the line defined by the point (*x, y*) and *angle*.

Return value

None.

See also

Translate2D, Rotate2D, Scale2D

GetViewerName

Function

Gets the name of a viewer.

Syntax

BOOL GetViewerName (int hview, LPSTR name);

Remarks

GetViewerName gets the name string of the viewer *hview*.

Return value

On success, GetViewerName returns TRUE. On error, it returns FALSE.

See also

DisplayViewerName, SetViewerName

SetViewerName

Function

Sets the name of a viewer.

Syntax

BOOL SetViewerName (int hview, LPSTR name);

Remarks

SetViewerName sets the name string of the viewer *hview*.

Return value

On success, SetViewerName returns TRUE. On error, it returns FALSE.

See also

DisplayViewerName, GetViewerName

GetViewport

Function

Gets the position of a viewport.

Syntax

BOOL GetViewport (int hview, LPRECT port);

Remarks

GetViewport gets the viewer *hview*'s viewport position in display coordinates to *port*.

Return value

On success, GetViewport returns TRUE. On error, it returns FALSE.

See also

SetViewport

CreateLight

Function

Creates a light

Syntax

int CreateLight (LPSTR name, int type);

Remarks

CreateLight creates a light with given *name* and *type*. The available light type *type* is one of the following.

VL_POINTLIGHT	point light with rays in all directions
VL_DISTLIGHT	distant light with parallel rays
VL_SPOTLIGHT	spot light with restricted angle

Return value

CreateLight returns the id of the newly created light. It returns 0 if it fails to create the light.

See also

DeleteLight

DeleteLight

Function
Deletes a light.

Syntax
void DeleteLight (int light);

Remarks
DeleteLight deletes the *light*.

Return value
None.

See also
CreateLight

CopyLight

Function

Copies the setting of a light.

Syntax

BOOL CopyLight (int lightdst, int lightsrc);

Remarks

CopyLight copies the settings of lightsrc to lightdst.

Return value

CopyLight returns TRUE if successful. On error, it returns FALSE

See also

CreateLight, DeleteLight

SelectLight

Function

Selects a light.

Syntax

BOL SelectLight (int light);

Remarks

SelectLight selects *light* as the current light.

Return value

SelectLight returns TRUE if successful. On error it returns FALSE.

See also

CreateLight

SwitchLight

Function

Switchs a light.

Syntax

int SwitchLight (int light, int action);

Remarks

SwitchLight turns the *light* on or off. *action* is one of the following.

TRUE	turn on the light
FALSE	turn off the light
VL_INQUIRE	inquire the status

Return value

Previous light status.

See also

CreateLight

CreateLModel

Function

Creates a light model

Syntax

int CreateLModel (LPSTR name);

Remarks

CreateLModel creates a light model with given *name*.

Return value

Light model id. 0 if it fails.

See also

DeleteLModel

DeleteLModel

Function
Deletes a light model.

Syntax
void DeleteLModel (int lmodel);

Remarks
DeleteLModel deletes a light model.

Return value
None.

See also
CreateLModel

CopyLModel

Function

Copies the settings of a light model.

Syntax

BOOL CopyLModel (int lmodeldst, int lmodelsrc);

Remarks

CopyLModel copies the setting of lmodelsrc to lmodeldst.

Return value

CopyLModel returns TRUE if successful. On error it returns FALSE.

See also

CreateLModel, DeleteLModel

SelectLModel

Function

Selects a light model.

Syntax

int SelectLModel (int lmodel);

Remarks

SelectLModel selects *lmodel* as the current light model.

Return value

Previous light model.

See also

CreateLModel

CreateMaterial

Function

Creates a material

Syntax

int CreateMaterial (LPSTR name);

Remarks

CreateMaterial creates a material with given *name*.

Return value

Material id. 0 if it fails.

See also

DeleteMaterial

DeleteMaterial

Function

Deletes a material.

Syntax

void DeleteMaterial (int material);

Remarks

DeleteMaterial deletes the *material*.

Return value

None.

See also

CreateMaterial

CopyMaterial

Function

Copies the settings of a material.

Syntax

BOOL CopyMaterial (int materaldst, int materialsrc);

Remarks

CopyMaterial copies the settings of materialsrc to materialdst.

Return value

CopyMaterail returns TRUE if successful. On error it returns FALSE.

See also

CreateMaterial, DeleteMaterial

SelectMaterial

Function

Selects a material.

Syntax

int SelectMaterial (int materail);

Remarks

SelectMaterail selects *material* as the current material.

Return value

Previous material.

See also

CreateMaterial

ShadingOption

Function

Sets shading options.

Syntax

int ShadingOption (int lmid, int option, int value);

Remarks

ShadingOption sets a shading option. lmid is the id of the lighting model. The available options and their values are the following

VL_SHADINGMETHOD VL_WIREFRAME
VL_SOLIDFILL
VL_FLATSHADE
VL_PHONESHADE
VL_GOURAUDSHADE

VL_SHADINGMODEL VL_PHONEMODEL
VL_SPECULARMODEL

The following options take Boolean values
VL_LOCALVIEWER
VL_BACKFACEREMOVAL
VL_DEPTHBUFFER
VL_TWOSIDESHADE
VL_COUNTCLOCKWISE

Return value

Previous value of the option.

See also

ShadingParameter

ShadingParameter

Function

Sets shading parameters.

Syntax

BOOL ShadingParameter (int lmid, int parameter, BOOL inquire, VECTOR value);

Remarks

ShadingParameter sets or inquires shading parameters. *lmid* is the id of the lighting model. *inquire* is set to TRUE for inquiry of a shading parameter. *parameter* is one of the following
VL_ATTENUATION
VL_LIGHTLOCATION
VL_LIGHTDIRECTION

Return value

TRUE if successful. FALSE on error.

See also

ShadingOption

ShadingColor

Function

Sets shading colors.

Syntax

COLORREF ShadingColor (int id, int type, COLORREF color);

Remarks

ShadingColor sets various shading colors. *id* is the id of light, light model, or material. *type* is one of the following.
VL_BACKGROUNDCOLOR
VL_MATERIALDIFFUSE
VL_MATERIALAMBIENT
VL_MATERIALEMISSION
VL_MATERIALSPECULAR
VL_LIGHTCOLOR
VL_LIGHTAMBIENT

Return value

Previous color.

See also

ShadingFactor

ShadingFactor

Function

Sets shading factors.

Syntax

float ShadingFactor (int id, int type, float factor);

Remarks

ShadingFactor sets the intensity factors of various shading colors. *id* is the id of the material, light, or light model. type can be one of the following.
VL_AMBIENTREFLECT
VL_DIFFUSEREFLECT
VL_SPECULARREFLECT
VL_EMISSIONSTRENGTH
VL_SHININESS
VL_LIGHTINTENSITY
VL_AMBIENTATTRIB
VL_SPOTLIGHTANGLE
VL_SPOTLIGHTSPREAD
VL_GLOBEAMBIENT

Return value

Previous value.

See also

ShadingColor

BeginDoubleBuffer

Function

Starts double buffer mode.

Syntax

BOOL BeginDoubleBuffer (HDC FAR *phdc, int hview);

Remarks

BeginDoubleBuffer starts the double buffer mode for the viewer *hview*. *phdc* is a pointer to the handle of the device context used by the viewer. After calling this function, all drawing function calls to the viewer will be redirected to a buffer. The buffer can be displayed by calling UpdateDoubleBuffer.

Return Value

On success, BeginDoubleBuffer returns TRUE. On error, it returns FALSE.

See also

EndDoubleBuffer, UpdateDoubleBuffer

EndDoubleBuffer

Function

Ends double buffer mode.

Syntax

BOOL EndDoubleBuffer (HDC FAR *phdc, int hview);

Remarks

EndDoubelBuffer ends the double buffer mode and releases the memory allocated for the buffer.

Return Value

On success, EndDoubleBuffer returns TRUE. On error, it returns FALSE.

See also

BeginDoubleBuffer, UpdateDoubleBuffer

UpdateDoubleBuffer

Function

Displays the buffered image in the double buffer mode.

Syntax

BOOL UpdateDoubleBuffer (HDC hdc, int hview);

Remarks

UpdateDoubleBuffer displays the buffered image in the double buffer mode. The content of the buffer is copied to the actual device context.

Return Value

On success, UpdateDoubleBuffer returns TRUE. On error, it returns FALSE.

See also

BeginDoubleBuffer, EndDoubleBuffer

SetDepthBuffer

Function

Sets the depth buffer.

Syntax

BOOL SetDepthBuffer (int hview);

Remarks

SetDepthBuffer sets a depth buffer (z-buffer) for the viewer *hview*.

Return value

SetDepthBuffer returns TRUE if successful. On error, it returns FALSE.

See also

ClearDepthBuffer

ClearDepthBuffer

Function

Clears the depth buffer.

Syntax

void ClearDepthBuffer (WORD value);

Remarks

ClearDepthBuffer clears the depth buffer with the given *value*.

Return value

None.

See also

SetDepthBuffer

FreeDepthBuffer

Function

Frees depth buffer.

Syntax

BOOL FreeDepthBuffer (int hview);

Remarks

FreeDepthBuffer frees the depth buffer for the viewer *hview*.

Return value

FreeDepthBuffer returns TRUE if successful. On error it returns FALSE

See also

SetDepthBuffer

SetPoint2D

Function

Sets a 2D point.

Syntax

void SetPoint2D (LPPOINT2D point, float x, float y);

Remarks

SetPoint2D assigns the value of the 2D *point* with coordinates *x* and *y*.

Return value

None.

See also

SetPoint2H

SetPoint2H

Function

Sets a 2D homogeneous point.

Syntax

void SetPoint2H (LPPOINT2H point, float x, float y, float w);

Remarks

SetPoint2H assigns the value of the *point* with the homogeneous coordinates *x*, *y*, *w*.

Return value

None.

See also

SetPoint2D

SetPoint3D

Function

Sets a 3D point.

Syntax

void SetPoint3D (LPPOINT3D point, float x, float y, float z);

Remarks

SetPoint3D assigns the value of the 3D point with coordinates *x*, *y*, *z*.

Return value

None.

See also

SetPoint3H

SetPoint3H

Function

Sets a 3D homogeneous point.

Syntax

void SetPoint3H (LPPOINT3H point, float x, float y, float z, float w);

Remarks

SetPoint3H assigns the value of the *point* with the homogeneous coordinates *x*, *y*, *z*, *w*.

Return value

None.

See also

SetPoint3D

MoveTo2D

Function

Moves to a new display position.

Syntax

void MoveTo2D (HDC hdc, float x, float y);

Remarks

MoveTo2D moves the current 2D display position to (*x, y*) in the current viewer.

Return value

None.

See also

LineTo2D

RMoveTo2D

Function

Moves the current display point relatively.

Syntax

void RMoveTo2D (HDC hdc, float dx, float dy);

Remarks

RMoveTo2D moves the display position by increments *dx* and *dy*.

Return value

None.

See also

MoveTo2D

LineTo2D

Function

Draws a 2D line to a new position.

Syntax

void LineTo2D (HDC hdc, float x, float y);

Remarks

LineTo2D draws a 2D line from the current 2D display position to (*x, y*) in the current viewer with the current pen.

Return value

None.

See also

MoveTo2D

RLineTo2D

Function

Draws a line relatively.

Syntax

void RLineTo2D (HDC hdc, float dx, float dy);

Remarks

RLineTo2D draws a line from the current display position to the point with increments *dx* and *dy*.

Return value

None.

See also

LineTo2D

Line2D

Function

Draws a 2D line segment.

Syntax

void Line2D (HDC hdc, float x1, float y1, float x2, float y2);

Remarks

Line2D draws a 2D line from (*x1, y1*) to (*x2, y2*) in the current 2D viewer with the current pen.

Return value

None.

See also

LineTo2D, MoveTo2D

RLine2D

Function

Draws a line.

Syntax

void RLine2D (HDC hdc, float x, float y, float dx, float dy);

Remarks

RLine2D draws a line from the point (*x, y*) to (*x+dx, y+dy*) with the current pen.

Return value

None.

See also

Line2D

MoveTo2H

Function

Moves the current 2D display position.

Syntax

void MoveTo2H (HDC hdc, float x, float y, float w);

Remarks

MoveTo2H moves the 2D diaplay position to the point with homogeneous coordinates (*x, y, w*).

Return value

None.

See also

MoveTo2D

LineTo2H

Function

Draws a line.

Syntax

void LineTo2H (HDC hdc, float x, float y, float w);

Remarks

LineTo2H draws a line from the current display position to the point given by the homogeneous coordinates (*x, y ,w*);

Return value

None.

See also

LineTo2D

Line2H

Function

Draws a line.

Syntax

void Line2H (HDC hdc, float x1, float y1, float w1, float x2, float y2, float w2);

Remarks

Line2H draws a line from point with homogeneous coordinate (*x1, y1, w1*) to (*x2, y2, w2*).

Return value

None.

See also

Line2D

Polyline2D

Function

Draws a 2D polyline.

Syntax

void Polyline2D (HDC hdc, int type, LPCOORD points, short n);

Remarks

Polyline2D draws a 2D polyline defined by *points* of coordinate type *type* in the current 2D viewer with current pen. *count* is the number of vertices.

Return value

None.

See also

Polygon2D

ClosedPolyline2D

Function

Draws a closed polyline.

Syntax

void ClosedPolyline2D (HDC hdc, int type, LPCOORD points, int count);

Remarks

ClosedPolyline2D draws a closed polyline. The vertices of the polyline is given by *points*. *type* specifies the coordinate type of *points* and the number of vertices is *count*.

Return value

None.

See also

Polyline2D

Polygon2D

Function

Draws a 2D polygon.

Syntax

void Polygon2D (HDC hdc, int type, LPCOORD points, int count);

Remarks

Polygon2D draws a 2D polygon defined by *points* of coordinate type *type* in the current 2D viewer with current pen for edges and current brush for interior. *count* is the number of points.

Return value

None.

See also

Polyline2D

PolyPolygon2D

Function

Draws a polypolygon.

Syntax

void PolyPolygon2D (HDC hdc, int type, LPCOORD points, LPINT polycount, int count);

Remarks

PolyPolygon2D draws a polypolygon. The vertices are given by *points* and their coordinate type is given by type.

Return value

None.

See also

Polygon2D

Mark2D

Function

Draws a 2D mark.

Syntax

void Mark2D (HDC hdc, real x, real y, int hsize, int vsize, int marktype);

Remarks

Mark2D draws a mark of given *marktype* at (*x, y*) with horizontal size *hsize* and vertical size *vsize*.
marktype is one of the following.
VL_NULLMARK
VL_CIRCLEMARK
VL_CROSSMARK
VL_XMARK
VL_TRIANGLEMARK
VL_BOXMARK
VL_DIAMONDMARK
VL_HEXAGONMARK

Return value

None.

See also

PolyMark2D

PolyMark2D

Function

Draws a sequence of marks.

Syntax

void PolyMark2D (HDC hdc, int type, LPCOORD point, int n, int hsize, int vsize, int marktype);

Remarks

PolyMark2D draws a sequence of *n* marks of *marktype* at *points* with horizontal size *hsize* and vertical size *vsize*. *marktype* is one of the following.

VL_NULLMARK
VL_CIRCLEMARK
VL_CROSSMARK
VL_XMARK
VL_TRIANGLEMARK
VL_BOXMARK
VL_DIAMONDMARK
VL_HEXAGONMARK

Return value

None.

See also

Mark2D

Arrow2D

Function

Draws an arrow.

Syntax

void Arrow2D (HDC hdc, float x, float y, float u, float v, float r, float l, float w, int type);

Remarks

Arrow2D draws a 2D arrow of length *r* from (*x, y*) in the direction (*u, v*). *l* and *w* are the length and width of the arrow head. The arrow *type* is one of the following.
VL_NULLARROW
VL_OPENARROW
VL_CLOSEDARROW

Return value

None.

See also

Mark2D

MarkPosition2D

Function

Draws a mark.

Syntax

void MarkPosition2D (HDC hdc, float x, float y, float size, int type);

Remarks

MarkPosition2D draws a mark at (*x, y*). The *size* is specified in terms of object coordinates. *type* defines the type of marks which can take the following values.

VL_CROSSHAIR	cross hair mark
VL_ORIGIN	two arrows from the origin

Return value

None.

See also

Mark2D

Net2D

Function

Draws a 2D net.

Syntax

void Net2D (HDC hdc, int type, LPCOORD points, int m, int n);

Remarks

Net2D draws a *m* by *n* 2D net with the vertices pointed by *points*. The coordinate type of point is *type*.

Return value

None.

See also

Polygon2D

Rectangle2D

Function

Draws a 2D rectangle.

Syntax

void Rectangle2D (HDC hdc, float x1, float y1, float x2, float y2);

Remarks

Rectangle2D draws a 2D rectangle defined by two corner points (*x1, y1*) and (*x2, y2*) in the current 2D viewer with current pen for edge and current brush for interior.

Return value

None.

See also

Polygon2D

Disk2D

Function

Draws a 2D elliptic disk.

Syntax

void Disk2D (HDC hdc, float x, float y, float angle, float a, float b);

Remarks

Disk2D draws a 2D elliptic disk with center (*x, y*), the half major axis *a* and the half minor axis *b* in the current 2D viewer with current pen for edge and current brush for interior. The disk is rotated by *angle*.

Return value

None.

See also

Arc2D

Arc2D

Function

Draws a 2D elliptic arc.

Syntax

void Arc2D (HDC hdc, float x, float y, float angle, float a, float b, float start, float end);

Remarks

Arc2D draws a 2D arc in the current 2D viewer with the current pen color. (*x,y*) is the center of the ellipse and *angle* is the angle of the major axis. *a* and *b* are the half lengths of the major and minor axes. The arc is drawn from the angle *start* to *end*. All angles are measured in degrees.

Return Value

None.

See also

Disk2D

Wedge2D

Function

Draws a 2D elliptic wedge.

Syntax

void Wedge2D (HDC hdc, float x, float y, float angle, float a, float b, float start, float end);

Remarks

Wedge2D draws a 2D elliptic wedge (arc with the two radial lines) in the current 2D viewer with the current pen color for the edges and the current brush color for the interior. (*x,y*) is the center of the ellipse, *angle* is the angle of the major axis of the ellipse, and *a* and *b* are the half lengths of the major and minor axes. The arc are drawn from angle *start* to *end* measured in degrees.

Return Value

None.

See also

Bow2D

Bow2D

Function

Draws a 2D elliptic bow.

Syntax

void Bow2D (HDC hdc, float x, float y, float angle, float a, float b, float start, float b);

Remarks

Bow2D draws a 2D elliptic bow (arc with the chord) in the current 2D viewer with current pen for edge and current brush for interior. (*x,y*) is the center of the ellipse, *angle* is the angle of the major axis of the ellipse, and *a* and *b* are the half lengths of the major and minor axes. The arc are drawn from angle *start* to *end* measured in degrees.

Return value

None.

See also

Wedge2D

Ngon2D

Function

Draws a 2D n sided polygon.

Syntax

void Ngon2D (HDC hdc, float x, float y, float angle, float a, float b, short n);

Remarks

Ngon2D draws a 2D *n* sided polygon in the current 2D viewer with current pen for edge and current brush for interior. The polygon can be inscribed in an ellipse and the vertices form equal angles about the center. (*x, y*) is the center, *angle* is the initial angle, and *a* and *b* are the half lengths of the major and minor axes. .

Return value

None.

See also

Polygon2D

Star2D

Function

Draws a 2D n point star.

Syntax

void Star2D (HDC hdc, float x, float y, float angle, float a, float b, int n);

Remarks

Star2D draws a 2D *n* point star in the current 2D viewer with current pen for edge and current brush for interior. The star can be inscribed in an ellipse and the vertices form equal angles about the center. (*x, y*) is the center, *angle* is the initial angle, and *a* and *b* are the half lengths of the major and minor axes.

Return value

None.

See also

Polygon2D

Flower2D

Function

Draws a 2D n leaf flower.

Syntax

void Flower2D (HDC hdc, float x, float y, float angle, float a, float b, int n, float ratio);

Remarks

Flower2D draws a 2D *n* leaf flower in the current 2D viewer with current pen for edge and current brush for interior. The vertices of the flower lie on two ellipses. (*x, y*) is the center, *angle* is the initial angle, and *a* and *b* are the half lengths of the major and minor axes of an ellipse. The other ellipse is obtained by scaling of *ratio*.

Return value

None.

See also

Star2D

MoveTo3D

Function

Moves current 3D display position.

Syntax

void MoveTo3D (HDC hdc, float x, float y, float z);

Remarks

MoveTo3D moves current 3D display position to (*x, y, z*) in the current 3D viewer.

Return value

None.

See also

LineTo3D

RMoveTo3D

Function

Moves display position relatively.

Syntax

void RMoveTo3D (HDC hdc, float dx, float dy, float dz);

Remarks

RMoveTo3D moves the 3D display position relative to the current position by the amount *dx*, *dy*, *dz*.

Return value

None.

See also

MoveTo3D

LineTo3D

Function

Drawsa 3D line to a new position.

Syntax

void LineTo3D (HDC hdc, float x, float y, float z);

Remarks

LineTo3D draws a 3D line from the current display position to (*x, y, z*) in the current viewer with current pen.

Return value

None.

See also

MovoTo3D

RLineTo3D

Function

Draws a line segment.

Syntax

void RLineTo3D (HDC hdc, float dx, float dy, float dz);

Remarks

RLineTo3D draws a line segment from the current position to the point with increments *dx, dy, dz*.

Return value

None.

See also

LineTo3D

Line3D

Function

Draws a 3D line segment.

Syntax

void Line3D (HDC hdc, float x1, float y1, float z1, float x2, float y2, float z2);

Remarks

Line3D draws a 3D line from the point (*x1, y1, z1*) to (*x2, y2, z2*) in the current 3D viewer with current pen.

Return value

None.

See also

LineTo3D, MovoTo3D

RLine3D

Function

Draws a line.

Syntax

void RLine3D (HDC hdc, float x, float y, float z, float dx, float dy, float dz);

Remarks

RLine3D draws a line from the point (*x, y, z*) to (*x+dx, y+dy, z+dz*).

Return value

None.

See also

Line3D

MoveTo3H

Function

Moves the 3D display position.

Syntax

void MoveTo3H (HDC hdc, float x, float y, float z, float w);

Remarks

MoveTo3H moves the display position to the point with homogeneous coordinates (*x, y, z, w*).

Return value

None.

See also

MoveTo3D

LineTo3H

Function

Draws a line segment.

Syntax

void LineTo3H (HDC hdc, float x, float y, float z, float w);

Remarks

LineTo3H draws a line from the current display position to the point with homogeneous coordinates (*x, y, z, w*).

Return value

None.

See also

LineTo3D

Line3H

Function

Draws a line.

Syntax

void Line3H (HDC hdc, float x1, float y1, float z1, float w1, float x2, float y2, float z2, float w2);

Remarks

Line3H draws a line from point with homogeneous coordinate (*x1, y1, z1, w1*) to (*x2, y2, z2, w2*).

Return value

None.

See also

Line3D

MarkPosition3D

Function

Draws a 3D position mark.

Syntax

void MarkPosition3D (HDC hdc, float x, float y, float z, float size, int marktype);

Remarks

MarkPosition3D draws a 3D position mark of *size* at point (*x, y, z*) in the current 3D viewer with red, green, and blue for the three axes. *marktype* is one of the following.
VL_CROSSHAIR
VL_ORIGIN

Return value

None.

See also

Mark3D

Polyline3D

Function

Draws a 3D polyline.

Syntax

void Polyline3D (HDC hdc, int type, LPCOORD points, int count);

Remarks

Polyline3D draws a 3D polyline defined by *points* of coordinate type *type* in the current 3D viewer with current pen. *count* is the number of vertices.

Return value

None.

See also

Polygon3D

ClosedPolyline3D

Function

Draws a closed polyline.

Syntax

void ClosedPolyline3D (HDC hdc, int type, LPCOORD points, int count);

Remarks

ClosedPolyline3D draws a closed polyline. The vertices are pointed by *points*. The coordinate type of points is *type* and the number of vertices is *count*. The polyline is closed automatically but the interior is not filled.

Return value

None.

See also

Polyline3D

Polygon3D

Function

Draws a 3D polygon.

Syntax

void Polygon3D (HDC hdc, LPPOINT3D point, short n);

Remarks

Polygon3D draws a 3D polyline defined by *points* of coordinate type *type* in the current 3D viewer with current pen the edges and current brush for the interior. *count* is the number of vertices.

Return value

None.

See also

Polyline3D

PolyPolygon3D

Function

Draws a polypolygon.

Syntax

void PolyPolygon3D (HDC hdc, int type, LPCOORD points, LPINT polycount, int count);

Remarks

PolyPolygon3D draws a sequence of 3D polygons. The vertices are pointed by *points*. The coordinate type of points is type. The numbers of vertices in the polygons are in *polycount* and the number of polygons is *count*.

Return value

None.

See also

Polygon3D

Rectangle3D

Function

Draws a Rectangle.

Syntax

void Rectangle3D (HDC hdc, float x1, float y1, float x2, float y2);

Remarks

Rectangle3D draws a rectangle defined by two corner points (*x1, y1*) and (*x2, y2*) in the current 3D viewer with current pen for the edge and current brush for the interior.

Return value

None.

See also

Polygon3D

Mark3D

Function

Draws a 3D mark.

Syntax

void Mark3D (HDC hdc, float x, float y, float z, int hsize, int vsize, int marktype);

Remarks

Mark3D draws a mark of given *marktype* at (*x, y*) with horizontal size *hsize* and vertical size *vsize* that are given in screen coordinates. *marktype* is one of the following.

VL_NULLMARK
VL_CIRCLEMARK
VL_CROSSMARK
VL_XMARK
VL_TRIANGLEMARK
VL_BOXMARK
VL_DIAMONDMARK
VL_HEXAGONMARK

Return value

None.

See also

PolyMark3D

PolyMark3D

Function

Draws a sequence of marks.

Syntax

void PolyMark3D (HDC hdc, int type, LPCOORD point, int n, int hsize, int vsize, int head);

Remarks

PolyMark3D draws a sequence of *n* marks of *marktype* at *points* with horizontal size *hsize* and vertical size *vsize*. *marktype* is one of the following.
VL_NULLMARK
VL_CIRCLEMARK
VL_CROSSMARK
VL_XMARK
VL_TRIANGLEMARK
VL_BOXMARK
VL_DIAMONDMARK
VL_HEXAGONMARK

Return value

None.

See also

Mark3D

Arrow3D

Function

Draws an arrow.

Syntax

void Arrow3D (HDC hdc, float x, float y, float z, float u, float v, float w, float r, float l, float w, int type);

Remarks

Arrow3D draws an arrow of length *r* from (*x, y, z*) in the direction (*u, v, w*). *l* and *w* are the length and width of the arrow head. The arrow *type* is one of the following.
VL_NULLARROW
VL_OPENARROW
VL_CLOSEDARROW

Return value

None.

See also

Mark3D

Net3D

Function

Draws a 3D net.

Syntax

void Net3D (HDC hdc, int type, LPCOORD points, int m, int n);

Remarks

Net3D draws a *m* by *n* net with the vertices pointed by *points*. The coordinate type of points is *type*.

Return value

None.

See also

Polygon3D

Wedge3D

Function

Draws an elliptic wedge.

Syntax

void Wedge3D (HDC hdc, float x, float y, float angle, float a, float b, float start, float end);

Remarks

Wedge3D draws an elliptic wedge (arc with the two radial lines) in the current 3D viewer with the current pen color for the edges and the current brush color for the interior. (*x,y*) is the center of the ellipse, *angle* is the angle of the major axis of the ellipse, and *a* and *b* are the half lengths of the major and minor axes. The arc are drawn from angle *start* to *end* measured in degrees.

Return Value

None.

See also

Bow3D

Bow3D

Function

Draws an elliptic bow.

Syntax

void Bow3D (HDC hdc, float x, float y, float angle, float a, float b, float start, float end);

Remarks

Bow3D draws a 3D elliptic bow (arc with the chord) in the current 3D viewer with current pen for edge and current brush for interior. (*x,y*) is the center of the ellipse, *angle* is the angle of the major axis of the ellipse, and *a* and *b* are the half lengths of the major and minor axes. The arc are drawn from angle *start* to *end* measured in degrees.

Return value

None.

See also

Wedge3D

Ring

Function

Draws a part of ring.

Syntax

void Ring (HDC hdc, float rtop, float rbot, float thick, float h, float a, float b, float ratio);

Remarks

Ring draws a ring. *rtop* and *rbot* are the radii of the top and bottom cirlces. *h* is the height and *thick* is the thickness of the wall. *a* and *b* are the start and end angles. *ratio* defines the scaling ratio of y direction over x direction.

Return value

None.

See also

Tube

Tube

Function

Draws a tube.

Syntax

void Tube (HDC hdc, float rtop, float rbot, float thick, float h);

Remarks

Tube draws a tube which is a special ring consisting of full circles. *rtop* and *rbot* are the radii of the top and bottom circles. *thick* is the thickness of the tube wall and *h* is the height.

Return value

None.

See also

Ring

Prism

Function

Draws a 3D prism.

Syntax

void Prism (HDC hdc, LPPOINT3D base, LPPOINT3D top, int n);

Remarks

Prism draws a 3D prism of defined by the *n* points *base* and *n* points *top*.

Return value

None.

See also

Pyramid

Pyramid

Function

Draws a pyramid.

Syntax

void Pyramid (HDC hdc, LPPOINT3D base, int n, LPPOINT3D tip);

Remarks

Pyramid draws a pyramid in the current 3D viewer. The apex is specified by *tip*. The *n* base vertices is in *base*.

Return Value

None.

See also

Prism

Star3D

Function

Draw a n point star.

Syntax

void Star3D (HDC hdc, float x, float y, float angle, float a, float b, short n);

Remarks

Star3D draws an *n* point star on the xy plane in the current 3D viewer with current pen for edge and current brush for interior. The star can be inscribed in an ellipse and the vertices form equal angles about the center. (*x, y*) is the center, *angle* is the initial angle, and *a* and *b* are the half lengths of the major and minor axes.

Return value

None.

See also

Flower3D

Flower3D

Function

Draw a 3D n point flower.

Syntax

void Flower3D (HDC hdc, float x, float y, float z, float h, float r1, float r2, short n);

Remarks

Flower3D draws a *n* leaf flower on the xy plane in the current 3D viewer with current pen for edge and current brush for interior. The vertices of the flower lie on two ellipses. (*x, y*) is the center, *angle* is the initial angle, and *a* and *b* are the half lengths of the major and minor axes of an ellipse. The other ellipse is obtained by scaling of *ratio*.

Return value

None.

See also

Star3D

Cube

Function

Draws a 3D rectangular box.

Syntax

void Cube (HDC hdc, float w, float l, float h);

Remarks

Cube draws a 3D rectangular box defined by width *w*, length *l*, and height *h*.

Return value

None.

See also

Rectangle3D

Sphere

Function

Draws a sphere.

Syntax

void Sphere (HDC hdc, float r);

Remarks

Sphere draws a sphere of radius *r*.

Return value

None.

See also

Cylinder, Cone

Cone

Function

Draws a cone.

Syntax

void Cone (HDC hdc, float a, float b, float h);

Remarks

Cone draws a vertical elliptic cone defined by the half lengths of major and minor axes *a* and *b* and the height *h*.

Return value

None.

See also

Cylinder

Cylinder

Function

Draws a cylinder.

Syntax

void Cylinder (HDC hdc, float a, float b, float h);

Remarks

Cylinder draws a vertical elliptic cylinder defined by the half lengths of the major and minor axes *a* and *b* and the height *h*.

Return value

None.

See also

Cone

ShadePolygon

Function

Draws a polygon with shading.

Syntax

BOOL ShadePolygon (HDC hdc, VECTOR normal, int type, LPCOORD vertices, int count);

Remarks

ShadePolygon draws a polygon in the current 3D viewer with shading. The polygon is defined by *count vertices* of coordinate type *type*. *normal* is the normal of the polygon for shading. If *normal* is NULL, the polygon normal will be calculated.

Return value

ShadePolygon returns TRUE if successful. On error it returns FALSE.

See also

ShadePolyPolygon

ShadePolyPolygon

Function

Draws a polypolygon with shading.

Syntax

BOOL ShadePolyPolygon (HDC hdc, VECTOR normal, int type, LPCOORD vertices, LPINT polycount, int count);

Remarks

ShadePolyPolygon draws a polypolygon in the current 3D viewer with shading. The *count* polygons are defined by *polycount vertices* of coordinate type *type*. *normal* is the normal of the polygon for shading. If *normal* is NULL, the polygon normal will be calculated.

Return value

ShadePolyPolygon returns TRUE if successful. On error it returns FALSE.

See also

ShadePolygon

BezierCurve2D

Function

Draws a 2D Bezier curve.

Syntax

void BezierCurve2D (HDC hdc, int type, LPCOORD cp);

Remarks

BezierCurve2D draws a Bezier curve in the current 2D viewer. The curve is specified by four control points *cp* of corrdinate type *type*.

Return value

None.

See also

BSplineCurve2D, HermitCurve2D, NURBSCurve2D

HermitCurve2D

Function

Draws a 2D Hermit curve.

Syntax

void HermitCurve2D (HDC hdc, int type, LPCOORD cp);

Remarks

HermitCurve2D draws a Hermit curve in the current 2D viewer. The curve is specified by two control points and two tangent vectors in *cp* of corrdinate type *type*.

Return value

None.

See also

BezierCurve2D, BSplineCurve2D, NURBSCurve2D

BSplineCurve2D

Function

Draws a 2D uniform non-rational B-Spline curve.

Syntax

void BSplineCurve2D (HDC hdc, int type, LPCOORD cp, int n);

Remarks

BezierCurve2D draws a unform non-rational B-Spline curve in the current 2D viewer. The curve is specified by *n* control points *cp* of corrdinate type *type*. The first and the last knots of the spline are of multiplicity 3 and all othe knots are simple and uniformly spaced.

Return value

None.

See also

BezierCurve2D, HermitCurve2D, NURBSCurve2D

NURBSCurve2D

Function

Draws a 2D NURBS curve.

Syntax

void NURBSCurve2D(HDC hdc, int type, LPCOORD2D cp, int n, float FAR *knots);

Remarks

NURBSCurve2D draws a non-uniform rational B-spline (NURBS) curve in the current 2D viewer. The curve is specified by *n* control points *cp* of corrdinate type *type* and n+2 *knots*.

Return value

None.

See also

BezierCurve2D, BSplineCurve2D, HermitCurve2D

BSplineCurveClosed2D

Function

Draws a 2D closed uniform non-rational B-Spline curve.

Syntax

void BSplineCurveClosed2D (HDC hdc, int type, LPCOORD cp, int n);

Remarks

BezierCurveClosed2D draws a closed unform non-rational B-Spline curve in the current 2D viewer. The curve is specified by *n* control points *cp* of corrdinate type *type*. The last control point is considered to be followed by the first control point to form a closed curve. All knots are simple and uniformly spaced.

Return value

None.

See also

NURBSCurveClosed2D

NURBSCurveClosed2D

Function

Draws a closed 2D NURBS curve.

Syntax

void NURBSCurveClosed2D(HDC hdc, int type, LPCOORD cp, int n, float FAR *knots);

Remarks

NURBSCurveClosed2D draws a closed non-uniform rational B-spline (NURBS) curve in the current 2D viewer. The curve is specified by *n* control points *cp* of corrdinate type *type* and *n+1* knots.

Return value

None.

See also

BSplineCurveClosed2D

CatmullRomSpline2D

Function

Draws a Catmull Rom spline curve.

Syntax

void CatmullRomSpline2D (HDC hdc, int type, LPCOORD cp, int n);

Remarks

CatmullRomSpline2D draws a Catmull Rom spline curve in the current 2D viewer. The curve is defined by *n* control points *cp* of corrdinate type *type*.

Return value

None.

See also

BSplineCurve2D

BezierCurve3D

Function

Draws a 3D Bezier curve.

Syntax

void BezierCurve3D(HDC hdc, int type, LPCOORD cp);

Remarks

BezierCurve3D draws a Bezier curve in the current 3D viewer. The curve is specified by four control points *cp* of corrdinate type *type*.

Return value

None.

See also

BSplineCurve3D, HermitCurve3D, NURBSCurve3D

HermitCurve3D

Function

Draws a 3D Hermit curve.

Syntax

void HermitCurve3D (HDC hdc, int type, LPCOORD cp);

Remarks

HermitCurve3D draws a Hermit curve in the current 3D viewer. The curve is specified by two control points and two tangent vectors *cp* of corrdinate type *type*.

Return value

None.

See also

BezierCurve3D, BSplineCurve3D, NURBSCurve3D

BSplineCurve3D

Function

Draws a 3D uniform non-rational B-Spline curve.

Syntax

void BSplineCurve3D (HDC hdc, int type, LPCOORD cp, int n);

Remarks

BezierCurve3D draws a unform non-rational B-Spline curve in the current 3D viewer. The curve is specified by *n* control points *cp* of corrdinate type *type*. The first and the last knots are of multiplicity 3 and all othe knots are simple and uniformly spaced.

Return value

None.

See also

BezierCurve3D, HermitCurve3D, NURBSCurve3D

NURBSCurve3D

Function

Draws a 3D NURBS curve.

Syntax

void NURBSCurve3D (HDC hdc, int type, LPCOORD cp, int n, float FAR *knots);

Remarks

NURBSCurve3D draws a non-uniform rational B-spline (NURBS) curve in the current 3D viewer. The curve is specified by *n* control points *cp* of corrdinate type *type* and *n+2* knots.

Return value

None.

See also

BezierCurve3D, BSplineCurve2D, HermitCurve3D

BezierSurface

Function

Draws a Bezier surface.

Syntax

BOOL BezierSurface (HDC hdc, int type, LPCOORD cp, int ns, int nt);

Remarks

BezierSurface draws a Bezier surface in the current 3D viewer. The surface is specified by an array of 4 by 4 control points *cp* of corrdinate type *type*. The surface is drawn with *ns* sections in s direction and *nt* sections in the t direction.

Return value

TRUE if successful and FALSE if fails.

See also

BSplineSurface, HermitSurface, NURBSSurface

HermitSurface

Function

Draws a Hermit surface.

Syntax

BOOL HermitSurface(HDC hdc, int type, LPCOORD cp, int ns, int nt);

Remarks

HermitSurface draws a Hermit surface in the current 3D viewer. The surface is specified by 4 by 4 control points *cp* of corrdinate type *type*. The surface is drawn with *ns* sections in s direction and *nt* sections in the t direction.

Return value

TRUE if successful and FALSE if fails.

See also

BezierSurface, BSplineSurface, NURBSSurface

BSplineSurface

Function

Draws a uniform non-rational B-Spline surface.

Syntax

BOOL BSplineSurface(HDC hdc, int type, LPCOORD cp, int n1, int n2, int ns, int nt);

Remarks

BSplineSurface draws a unform non-rational B-Spline surface in the current 3D viewer. The surface is specified by *n1* by *n2* control points *cp* of corrdinate type *type* . The first and the last knots in each direction are of multiplicity 3 and all othe knots are simple and uniformly spaced. The surface is drawn with *ns* sections in s direction and *nt* sections in the t direction for each rectangular patch.

Return value

TRUE if successful and FALSE if fails.

See also

BezierSurface, HermitSurface, NURBSSurface

NURBSSurface

Function

Draws a NURBS surface.

Syntax

void NURBSSurface(HDC hdc, LPCOORD cp, float FAR *sknots, float FAR *tknots, int n1, int n2, int ns, int nt);

Remarks

NURBSSurface draws a non-uniform rational B-spline (NURBS) surface in the current 3D viewer. The surface is specified by *n1* by *n2* control points *cp* of corrdinate type *type* and with *n1+2 sknots* and *n2+2 tknots*. The surface is drawn with *ns* sections in s direction and *nt* sections in the t direction for each rectangular patch.

Return value

TRUE if successful and FALSE if fails.

See also

BezierSurface, BSplineSurface, HermitSurface

CoonsPatch

Function

Draws a Coons patch.

Syntax

BOOL CoonsPatch (HDC hdc, int type, LPCOORD cp, int nv, int nv);

Remarks

CoonsPatch draws a Coons patch in current 3D viewer. The surface is specified by 2(*nu*+*nv*) boundary control points *cp* of corrdinate type *type.*

Return value

TRUE if successful and FALSE if fails.

See also

BezierSurface, BSplineSurface

Ellipsoid

Function

Draws an ellipsoid

Syntax

BOOL Ellipsoid (HDC hdc, float a, float b, float c);

Remarks

Ellipsoid draws an ellipsoid in the current 3D viewer.

Return value

Ellipsoid returns TRUE if successful. On error it return FALSE.

See also

Sphere

HemiSphere

Function

Draws a hemisphere.

Syntax

BOOL HemiSphere (HDC hdc, float r, float h);

Remarks

HemiSphere draws a section of sphere in the current 3D viewer. *r* is the radius of the sphere and *h* is the height of the section.

Return value

HemiSphere returns TRUE if successful. On error it return FALSE.

See also

Sphere

SolidStar

Function

Draws a solid star.

Syntax

BOOL SolidStar (HDC hdc, int n, float a, float b, float h);

Remarks

SolidStar draws a solid *n* star in the current 3D viewer. The half lengths of the major and minor axes of the ellipse are *a* and *b*. *h* is the height of the star.

Return value

SolidStar returns TRUE if successful. On error it return FALSE.

See also

SolidFlower

SolidFlower

Function

Draws a solid flower.

Syntax

BOOL SolidFlower (HDC hdc, int n, float ratio, float a, float b, float h);

Remarks

SolidFlower draws a *n* leaf solid flower in the current 3D viewer. The half lengths of the major and minor axes are *a* and *b*. The other ellipse is obtained by scaling of *ratio*. *h* is the height of the solid flower.

Return value

SolidFlower returns TRUE if successful. On error it return FALSE.

See also

SolidStar

Wedge

Function

Draws a solid wedge.

Syntax

BOOL Wedge (HDC hdc, float a, float b, float h, float start, float end);

Remarks

Wedge draws a solid elliptic wedge in the current 3D viewer. The half axes of the ellipse are *a* and *b*. *h* is the height of the wedge. The wedge is drawn from angle *start* to *end*.

Return value

Wedge returns TRUE if successful. On error it return FALSE.

See also

Wedge2D

Frustum

Function

Draws a solid frustum.

Syntax

BOOL Frustum (HDC hdc, float bw, float bl, float tw, float tl, float h);

Remarks

Frustum draws a frustum in the current 3D viewer. The bottom rectangle is *bw* by *bl* and the top rectangle *tw* by *tl*. *h* is the height.

Return value

Frustum returns TRUE if successful. On error it return FALSE.

See also

Cube

Ridge

Function

Draws a ridge.

Syntax

BOOL Ridge (HDC hdc, float w, float l, float h, float r);

Remarks

Ridge draws a ridge in the current 3D viewer. The bottom rectangle is *w* by *l*. *h* is the height and *r* is the length of the top ridge.

Return value

Ridge returns TRUE if successful. On error it return FALSE.

See also

Frustum

Parabola

Function

Draws a parabola.

Syntax

void Parabola (HDC hdc, float x1, float x2);

Remarks

Parabola draws a parabola curve in the current 2D viewer. *x1* and *x2* specity the start and end x values.

Return value

None.

See also

Hyperbola

Hyperbola

Function

Draws a hyperbola.

Syntax

void Hyperbola (HDC hdc, float y1, float y2);

Remarks

Hyperbola draws a branch of hyperbola curve in the current 2D viewer. *y1* and *y2* specify the start and end y values.

Return value

None.

See also

Parabola

OscillatoryWave

Function

Draws a sine wave.

Syntax

void OscillatoryWave (HDC hdc, float a, float b, float x1, float x2);

Remarks

OscillatoryWave draws a oscillaroty wave with equation *y=exp(-ax)sin (bx)* from *x1* to *x2* in the current 2D viewer.

Return value

None.

See also

Catenary

Catenary

Function

Draws a catenary.

Syntax

void Catenary (HDC hdc, real x1, real x2);

Remarks

Catenary draws a catenary in the current 2D viewer. *x1* and *x2* specify the start and end x values.

Return value

None.

See also

OscillatoryWave

Spiral2D

Function

Draws a 2D spiral.

Syntax

void Spiral2D (HDC hdc, float angle);

Remarks

Spiral2D draws a 2D spiral in the current 2D viewer. The spiral is drawn from angle 0 to *angle*.

Return value

None.

See also

Spiral3D

Cycloid

Function

Draws a cycloid.

Syntax

void Cycloid (HDC hdc, float angle);

Remarks

Cycloid draws a cycloid from 0 to *angle* in the current 2D viewer.

Return value

None.

See also

Epicycloid, Hypocycloid

Epicycloid

Function

Draws an epicycloid.

Syntax

void Epicycloid (HDC hdc, float a, float b);

Remarks

Epicycloid draws an epicycloid in the current 2D viewer. The equation is given by
x = (a+b) cos t - a cos ((a+b) t / a)
y = (a+b) sin t - a sin ((a+b) t / a)

Return value

None.

See also

Cycloid

Cardioid

Function

Draws a cardioid.

Syntax

void Cardioid (HDC hdc);

Remarks

Cardioid draws a cardioid in the current 2D viewer.

Return value

None.

See also

Cycloid

Hypocycloid

Function

Draws a hypocycloid.

Syntax

void Hypocycloid (HDC hdc, float a, float b);

Remarks

Hypocycloid draws a hypocycloid in the current 2D viewer. The equation is given by
x = (a-b) cos t + b cos ((a-b) t / b)
y = (a-b) sin t - b sin ((a-b) t / b)

Return value

None.

See also

Cycloid, Epicycloid

Lemniscate

Function

Draws a lemniscate.

Syntax

void Lemniscate (HDC hdc, float a);

Remarks

Lemniscate draws a lemniscate in the current 2D viewer. The polar equation is given by
r = a sqrt(2cos 2t)

Return value

None.

See also

Cardioid

Rose

Function

Draws a rose.

Syntax

void Rose (HDC hdc, int n, float a);

Remarks

Rose draws a rose curve in the current 2D viewer. The polar equation is given by
r = a cos nt

Return value

None.

See also

Lemniscate

Spring

Function

Draws a spring.

Syntax

void Spring (HDC hdc, int n, float radius, float height);

Remarks

Spring draws *n* rounds a spring of given *radius* and *height*.

Return value

None.

See also

Spiral3D

Spiral3D

Function

Draw a 3D spiral curve.

Syntax

Spiral3D (HDC hdc, float angle, float height);

Remarks

Spiral3D draws a 3D spiral curve of *height* from 0 to *angle* in the current 3D viewer.

Return value

None

See also

Spring

EllipticParaboloid

Function

Draws a elliptic paraboloid.

Syntax

void EllpticParaboloid (HDC hdc, float height, float count1, float count2);

Remarks

EllipticParaboloid draws an elliptic paraboloid *height* in the current 3D viewer. The surface is drawn with *count1* pieces in the circular sections and *count2* pieces in the vertical direction.

Return value

None.

See also

HyperbolicParaboloid

Hyperboloid1

Function

Draws a hyperboloid of one sheet.

Syntax

void Hyperboloid1 (HDC hdc, float z1, float z2, int count1, int count2);

Remarks

Hyperboloid1 draws a hyperboloid of one sheet in the current 3D viewer. The surface is drawn from *z1* to *z2* with *count1* pieces in the circular sections and *count2* pieces in the vertical direction.

Return value

None.

See also

Hyperboloid2

Hyperboloid2

Function

Draws a hyperboloid of two sheet.

Syntax

void Hyperboloid2 (HDC hdc, float height, int count1, int count2);

Remarks

Hyperboloid2 draws a hyperboloid of two sheets in the current 3D viewer. The surface is drawn with *count1* pieces in the circular sections and *count2* pieces in the vertical direction.

Return value

None.

See also

Hyperboloid1

HyperbolicParaboloid

Function

Draws a hyperbolic paraboloid.

Syntax

void HyperbolicParaboloid (HDC hdc, float x1, float x2, float y1, float y2, int count1, int count2);

Remarks

HyperbolicParaboloid draws a hyperbolic parabolid in the current 3D viewer. The surface is drawn from *x1* to *x2* and from *y1* to *y2* with *count1* by *count2* patchs.

Return value

None.

See also

EllipticParaboloid

ImageMap2D

Function

Maps an image to a 2D object.

Syntax

BOOL ImageMap2D (HDC hdc, HGLOBAL hdib, int type, LPCOORD vertices);

Remarks

ImageMap2D maps a bitmap image in the 2D viewer. *hdib* is a handle to a device independent image. The four corner points of the image are *vertices* of coordinate type.

Return value

ImageMap2D returns TRUE if successful. On error it returns FALSE.

See also

ImageMap3D

ImageMap3D

Function

Maps an image to a 3D object.

Syntax

BOOL ImageMap3D (HDC hdc, HGLOBAL hdib, int type, LPCOORD vertices);

Remarks

ImageMap3D maps a bitmap image in the 3D viewer. *hdib* is a handle to a device independent image. The four corner points of the image are *vertices* of coordinate type.

Return value

ImageMap3D returns TRUE if successful. On error it returns FALSE.

See also

ImageMap2D

SolidTexture

Function

Sets solid texture.

Syntax

BOOL SolidTexture (int texture);

Remarks

SolidTexture sets the solid textures to be rendered on the objects. The following *texture* are available.
VL_NULL
VL_WOODGRAIN
VL_MARBLE.
VL_GRANITE

Return value

SolidTexture returns TRUE if successful. On error it returns FALSE.

See also

ShadingOption

SetFont

Function

Sets current TrueType font.

Syntax

BOOL SetFont (const LPLOGFONT lplf);

Remarks

SetFont sets the current font to the logic font pointed by *lplf*.

Return value

SetFont returns TRUE is successful. On error it returns FALSE.

See also

DrawString

TextParameter

Function

Sets 3D font's characteristics.

Syntax

float TextParameter (int parameter, float value);

Remarks

SetFont3D sets font *parameters* to *value* for DrawString. *parameter* is one of the following.
VL_TEXT_TAB
VL_TEXT_HEIGHT
VL_TEXT_ASPECT
VL_TEXT_THICKNESS

Return value

Previous value of the parameter.

See also

DrawString

DrawString

Function

Draws a string of 3D text.

Syntax

BOOL DrawString (HDC hdc, LPSTR string, int mode);

Remarks

DrawString draws a *string* of text in given *mode*. The current TrueType font is used to rendering. *mode* is a combination of the following flags.
VL_2DTEXT
VL_SOLIDTEXT
VL_HORIZONTAL
VL_VERTICAL

Return value

DrawString returns TRUE if successful. On error it returns FALSE.

See also

SetFont