

# Visualib Help Index



[Overview](#)

[Programming Guide](#)

[Function Reference](#)

[Registration Information](#)

## **Registration Information**

[License Information](#)

[Warrenty](#)

[Registration Form](#)

## 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.

## Registration Form

Visual Tech Co.

P.O. Box 8735

Fort Wayne, IN 46898-8735

(219) 489-0235

---

Product	Quantity	Unit Price	Amount
Visualib for Windows _____ (Microsoft C version)	\$50.00	\$ _____	
Visualib for Windows _____ (Borland C++ version)	\$50.00	\$ _____	
Visualib for DOS _____ (Microsoft C version)	\$40.00	\$ _____	
Visualib 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 is a comprehensive state-of-the-art graphics library for the Microsoft Windows environment. It contains powerful and efficient functions to transform and display both 2D and 3D graphic objects. Visualib can be used with either Microsoft Windows Software Development Kit version 3.0 or Borland C++ version 2.0 and up.

Main features of Visualib includes:

2D and 3D viewing systems

Transformations and stack

Drawing functions

## **Viewing systems for 2D and 3D graphics**

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

## **Transformation functions and stack**

Visualib provides a sophisticated transformation mechanism to support virtually all types of graphics transformations. Visualib maintains a transformation stack which can be used in conjunction with the transformation functions to achieve flexible and efficient graphic effects.



## **Graphic object drawing functions**

Visualib supports a full set of common 2D and 3D drawing functions such as lines, polygons, ellipses, spheres, polyhedra, etc. Backface culling is implemented for 3D viewers. User may also select double buffer mode to achieve smooth animation effects.

Visualib also provides the powerful curve and surface drawing functions such as Bezier, Hermit curves, B-Spline, NURBS curves and surfaces.

Visualib greatly extends the capabilities of windows' GDI functions. Visualib uses float type for specifying coordinates and avoids the common integer overflow problem associated with the GDI functions. 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 uses the attributes such as colors, line width of the device context set by the GDI functions.

Visualib is the only graphics package for the Windows that delivers the power of high-end graphics work stations. For everyone interested in using graphics in the Windows, Visualib is an indispensable tool.

Whether you are developing a CAD application or simply want to draw a nice business chart, you will find that Visualib can save your time and money. Visualib will free the user from writing highly sophisticated and tedious graphics routines and obtain beautiful 2-D and 3-D graphics quickly.

#### About This Manual

Chapter II. GET STARTED provides a brief introduction on how to use Visualib in your windows programming.

Chapter III. INSIDE Visualib is a detailed explanation of all features of Visualib.

Chapter IV. Visualib FUNCTION REFERENCE is the alphabetical reference of all Visualib functions.

Appendix A. BIBLIOGRAPHY lists graphics books and research papers related to the features of Visualib.

Appendix B. COMMON QUESTIONS contains answers to some most commonly asked questions about Visualib.

## Getting Started

Visualib disk contains the following files:

README.1ST - read me first  
REGISTER.TXT - ASCII registration form  
VISUALIB.LIB - the main library file  
VISUALIB.H - the header file  
VISUALIB.HLP - on-line Windows help of Visualib  
VLIBDEMO.C - Visualib demo program source code  
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 illustrated the application of Visualib library to create beautiful graphics applications. It uses many features on Visualib and may serve as a template on using Visualib.

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 either `initialGraphics2D` or `InitialGraphics3D`. After the graphics system is initialized, you may create 2D or 3D viewers by calling `CreateViewer2D` or `CreateViewer3D`. 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 `ExitGraphics` function to exit the Visualib system.

Refer to 3.4 for a complete description of drawing functions.

The following is a very simple program segment that illustrates the general procedure of using Visualib library.

```
#include "windows.h"
#include "Visualib.h"

/* in the function initInstance */
    InitialGraphics3D(3, 100, 100);

/* in the function WindProc */
    case WM_CREAT:
        hVIEW = CreatViewer3D("sample viewer",100,20,100,100);

    case WM_PAINT:
```

# Visualib Programming Guide

[Getting Started](#)

[Initialization and Termination](#)

[Coordinate Systems](#)

[Viewer](#)

[Modeling Transformations](#)

[Drawing Functions](#)

## Visualib Initialization and Termination

The 2D and 3D systems contained in Visualib are completely independent. You may use one of them or both of them at any time. Depending on your choice of graphics systems, one or both of the following initialization functions should be called before using the Visualib systems.

InitialGraphics2D

InitialGraphics3D

The initialization function allocates and initializes necessary system variables. When calling InitialGraphics2D or InitialGraphics3D, you specify the maximum number of viewers, the maximum number of points for each object, and the maximum depth of the matrix stack. The initialization function allocates the internal memory based on the given information. If an parameter is set to 0, the default maximum value for that parameter will be used.

The following functions set the default maximum values.

Max2DViewer

Max3DViewer

Max2DMStack

Max3DMStack

If a parameter in the initialization function is not 0, then the given value overrides the default value.

To exit a Visualib graphics system, use the function

ExitGraphics

ExitGraphics will free the memory used by the Visualib system.

## Coordinate Systems

Visualib has three different coordinate systems that concern users.

The world coordinate system is the one that users deal with most often. A world coordinate system is a logical 2D or 3D coordinate in which most Visualib functions use to specify the 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

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

The coordinates used in the world coordinate system are the 2D and 3D homogenous coordinates. Visualib defines the following types.

```
typedef struct {float x,y,w} POINT2D;  
typedef struct {float x,y,z,w} POINT3D;
```

Therefore, 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 this 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.

To help users convert the regular nonhomogeneous coordinates to the format used by Visualib, Visualib provides the following functions:

AssignPoint2D  
AssignPoint3D



## **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.

## **Viewing Transformation**

The viewing transformation of a viewer defines the position of the eye related to the world coordinate system.

## **Projection Transformation**

Projection transformation of a viewer defines the view volume and the way it is mapped to the viewport. A 3D projection can be either perspective or orthogonal.

## Setup Viewer

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

Viewport  
Viewing Transformation  
Projection Transformation

The following functions create a 2D or 3D viewer and set its viewport and name:

CreateViewer2D  
CreateViewer3D

The viewport of a viewer can be changed by the following function.

SetViewport2D  
SetViewport3D

The name of a viewer is changed by the functions:

SetViewerName2D  
SetViewerName3D

The frame of a viewer can be displayed by the functions:

DisplayViewerFrame2D  
DisplayViewerFrame3D

The name of a viewer is displayed by the functions:

DisplayViewerName2D  
DisplayViewerName3D

The viewing transformation of a 2D viewer is set by

SetView2D

The projection of a 2D viewer are set by the following functions:

SetProjection2D

The following function combines the actions of SetView2D and SetProjection

SetWindow2D

The viewing transformation of a 3D viewer is set by the functions:

SetView3D  
SetPolarView3D

The projection of a 3D viewer is set by the following functions:

SetPerspective3D

### SetOrthogonal3D

To select a viewer for drawing, use the functions:

SelectViewer3D

SelectViewer2D

The content of a viewer is cleared by the functions:

ClearViewer2D

ClearViewer3D

The viewing transformations may be modified by the following functions

MoveViewer3D

MoveWorld3D

RotateViewer3D

RotateWorld3D

ZoomViewer3D

ZoomWorld3D

MoveViewer2D

RotateViewer2D

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:

Num2DViewer

Num3DViewer

ViewerPosition3D

ViewerOrientation3D

ViewerDirection3D

ViewerPosition2D

ViewerField3D

ViewerField2D

SetView3D defines the VRP, VPN, and VUP of the viewer. VRP is specified by the world coordinates VX, VY, VZ; VPN is specified by VRP and another point (RX, RY, RZ); VUP is determined by the twist angle, which is the angle of rotation about the VPN.

SetPolarView3D sets the viewer's VRP, VPN and VUP. (CX,CY,CZ) defines a reference center (not necessarily the origin) in the world coordinates. VRP is given by the polar coordinates (dist, Azim, Inc). Dist is the distance from VRP to the center. Azim is a rotation about y-axis and Inc is the angle of rotation about z-axis. VUP is again defined by the the twist angle.

SetPerspective3D defines a perspective projection according to the field of view angle of Fovy, Aspect ratio, Front and Back clipping panes.

SetOrthogonal3D defines an orthogonal projection according to the viewing box defined by the Left, Right, Bottom, Top, Front, and Back.

Visualib provides a rich set of viewing transformations to help users achieve various viewing effects.

View reference point (VRP) - A reference point in the view plane that defines the camera position.

View plane normal (VPN) - the direction normal to the view plane. Together with VRP, it defines the view plane and the direction of the projection.

View up vector (VUP) - the vector in the view plane that points to the up direction.

Field of view angle (Fovy) - the angle of the viewing pyramid in the y-direction. Together with the Aspect ratio, it defines the projection point.

Aspect ratio - the ratio the y size over the x size of the view pyramid.

Clip depth - the minimum and maximum clipping values in z direction. It defines the top and bottom of the viewing pyramid.

## Modeling Transformations and Matrix Stack

Transformations are important part of the graphics system. Visualib provides full support of all types of affine geometric transformations. Users may arbitrarily translate, scale, or rotate any object in any sequence.

[Rotate3D](#)  
[Translate3D](#)  
[Scale3D](#)  
[Rotate2D](#)  
[Translate2D](#)  
[Scale2D](#)

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

[PushMatrix2D](#)  
[PushMatrix3D](#)

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.

[PopMatrix2D](#)  
[PopMatrix3D](#)



## Drawing Functions

Visualib provides a full set of common 2D and 3D drawing functions.

[MoveTo2D](#)  
[LineTo2D](#)  
[DrawLine2D](#)  
[Polyline2D](#)  
[Polygon2D](#)  
[Rectangle2D](#)  
[Circle2D](#)  
[Ellipse2D](#)  
[Ngon2D](#)  
[NsideStar2D](#)  
[NsideFlower2D](#)  
[MoveTo3D](#)  
[LineTo3D](#)  
[DrawLine3D](#)  
[Polyline3D](#)  
[Polygon3D](#)  
[Rectangle3D](#)  
[Prism3D](#)  
[NsideStar3D](#)  
[NsideFlower3D](#)  
[Cube3D](#)  
[Sphere3D](#)  
[NsidePyramid3D](#)  
[Cone3D](#)  
[NsidePrism3D](#)  
[Cylinder3D](#)

Visualib also provides advanced curve and surface functions. Visualib supports cubic Bezier, Hermit, B-Spline, and NURBS curves and surfaces.

[BezierCurve2D](#)  
[HermitCurve2D](#)  
[BSplineCurve2D](#)  
[NURBSCurve2D](#)  
[BezierCurve3D](#)  
[HermitCurve3D](#)  
[BSplineCurve3D](#)  
[NURBSCurve3D](#)  
[BezierSurface3D](#)  
[HermitSurface3D](#)  
[BSplineSurface3D](#)  
[NURBSSurface3D](#)

NURBS (NonUniform Rational B-Spline) curves and surfaces have gained popularities in CAD/CAM because of their power and flexibility. NURBS has some distinctive advantages:

NURBS is invariant under perspective projections.

The continuity and smoothness of NURBS curve or surfaces can be controlled by the knots.

All conic sections and quadric surfaces can be represented by NURBS exactly.

## Visualib Function Reference

### A

[Arc2D](#)  
[AssignPoint2D](#)  
[AssignPoint3D](#)

### B

[BackfaceCulling](#)  
[BeginDoubleBuffer3D](#)  
[BeginDoubleBuffer2D](#)  
[BezierCurve2D](#)  
[BezierCurve3D](#)  
[BezierSurface3D](#)  
[BrushColor](#)  
[BSplineCurve2D](#)  
[BSplineCurve3D](#)  
[BSplineSurface3D](#)

### C

[Circle2D](#)  
[ClearViewer2D](#)  
[ClearViewer3D](#)  
[Cone3D](#)  
[CountClockwise](#)  
[CreateViewer2D](#)  
[CreateViewer3D](#)  
[Cube3D](#)  
[Cylinder3D](#)

### D

[DisplayViewerFrame2D](#)  
[DisplayViewerFrame3D](#)  
[DisplayViewerName2D](#)  
[DisplayViewerName3D](#)  
[Dodecahedron](#)  
[DrawLine2D](#)  
[DrawLine3D](#)

### E

[Ellipse2D](#)  
[EllipseArc2D](#)  
[EndDoubleBuffer3D](#)  
[EndDoubleBuffer2D](#)  
[ExitGraphics](#)

### G

[GetViewerName2D](#)  
[GetViewerName3D](#)  
[GetViewport2D](#)  
[GetViewport3D](#)

### H

HermitCurve2D  
HermitCurve3D

**I**

InitialGraphics2D  
InitialGraphics3D  
Icosahedron

**L**

LineTo2D  
LineTo3D

**M**

MarkPosition3D  
Max2DStack  
Max2DViewer  
Max3DStack  
Max3DViewer  
MoveTo2D  
MoveTo3D  
MoveViewer2D  
MoveViewer3D  
MoveWorld3D

**N**

Ngon2D  
NsideFlower2D  
NsideFlower3D      NsidePrism3D  
NsidePyramid3D  
NsideStar2D  
NsideStar3D  
Num2DViewer  
Num3DViewer  
NURBSCurve2D  
NURBSCurve3D  
NURBSSurface3D

**O**

Octahedron

**P**

PenColor  
Polygon2D  
Polygon3D  
Polyline2D  
Polyline3D  
PopMatrix2D  
PopMatrix3D  
Prism3D  
PushMatrix2D  
PushMatrix3D  
Pyramid3D

**R**

Rectangle2D

Rectangle3D  
Rotate2D  
Rotate3D  
RotateViewer2D  
RotateViewer3D  
RotateWorld3D

**S**

Scale2D  
Scale3D  
SelectViewer2D  
SelectViewer3D  
SetOrthogonal3D  
SetPerspective3D  
SetPolarView3D  
SetProjection2D  
SetView2D  
SetView3D  
SetViewerName2D  
SetViewerName3D  
SetViewport2D  
SetViewport3D  
SetWindow2D  
Sphere3D

**T**

Tetrahedron  
Translate2D  
Translate3D

**U**

UpdateBuffer3D  
UpdateBuffer2D

**V**

ViewerDirection3D  
ViewerField2D  
ViewerField3D  
ViewerOrientation3D  
ViewerPosition2D  
ViewerPosition3D

**W**

Wedge2D

**Z**

ZoomViewer3D  
ZoomWorld3D

### HermitSurface3D

- short SetWindow3D (HVIEW hview, float left, float right,  
float top, float bottom, float front, float back);
- void Revolution3D (HDC hdc, float x1, float y1, float z1,  
float x2, float y2, float z2,  
float start, float angle, LPPOINT3D vertex, short count);
- void Ball3D (HDC hdc, float x, float y, float z, float r);
- void Mark3D (HDC hdc, float x, float y, float z, LPSTR mark);

## **AssignPoint2D**

### **Function**

Assigns 2D homogeneous coordinate.

### **Syntax**

```
void AssignPoint2D(POINT2D *point, float x, float y);
```

### **Remarks**

AssignPoint2D sets the homogeneous coordinate in point by the x, y coordinate.

### **Return Value**

None.

### **See also**

[AssignPoint3D](#)

## **AssignPoint3D**

### **Function**

Assigns 3D homogeneous coordinate.

### **Syntax**

```
void AssignPoint3D (POINT3D *point, float x, float y, float z);
```

### **Remarks**

AssignPoint3D sets the homogeneous coordinate in point by the x, y, z coordinate.

### **Return Value**

None.

### **See also**

[AssignPoint2D](#)



## **BackfaceCulling**

### **Function**

Sets backface culling flag.

### **Syntax**

short BackfaceCulling (short flag);

### **Remarks**

BackfaceCulling sets the backface culling flag. If the flag is set to a nonzero value, the drawing functions will implement backface culling.

### **Return Value**

BackfaceCulling returns the previous value of backface culling flag.

### **See also**

[CountClockwise](#)

## CountClockwise

### Function

Sets the counter-clockwise flag.

### Syntax

short CountClockwise (short flag);

### Remarks

CountClockwise sets the counter-clockwise flag. The flag is used for backface culling to determine the direction of a polygon normal. If the flag is set to a nonzero value, the drawing functions will assume that a polygon is specified by the vertices in counter-clockwise order, i.e., the direction of the polygon normal is determined by the right-hand system.

### Return Value

CountClockwise returns the previous value of the counter-clockwise flag.

### See also

[BackfaceCulling](#)

## **BeginDoubleBuffer3D**

### **Function**

Starts double buffer mode.

### **Syntax**

```
short BeginDoubleBuffer3D (HDC *hpdc, HVIEW hview);
```

### **Remarks**

BeginDoubleBuffer3D starts the double buffer mode for the 3D viewer hview. hpdc 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 UpdateBuffer3D.

### **Return Value**

On success, BeginDoubleBuffer3D returns 0. On error, it returns a nonzero value.

### **See also**

[EndDoubleBuffer3D](#), [UpdateBuffer3D](#)

## **BeginDoubleBuffer2D**

### **Function**

Starts double buffer mode.

### **Syntax**

```
short BeginDoubleBuffer2D (HDC *hdc, HVIEW hview);
```

### **Remarks**

BeginDoubleBuffer2D starts the double buffer mode for the 2D viewer hview. hpdc 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 UpdateBuffer2D.

### **Return Value**

On success, BeginDoubleBuffer2D returns 0. On error, it returns a nonzero value.

### **See also**

[EndDoubleBuffer2D](#), [UpdateBuffer2D](#)

## **EndDoubleBuffer3D**

### **Function**

Ends double buffer mode.

### **Syntax**

short EndDoubleBuffer3D (HDC \*hdc, HVIEW hview);

### **Remarks**

EndDoubleBuffer3D ends the double buffer mode and releases the memory allocated for the buffer.

### **Return Value**

On success, EndDoubleBuffer3D returns 0. On error, it returns a nonzero value.

### **See also**

[BeginDoubleBuffer3D](#), [UpdateBuffer3D](#)

## **EndDoubleBuffer2D**

### **Function**

Ends double buffer mode.

### **Syntax**

short EndDoubleBuffer2D (HDC \*hdc, HVIEW hview);

### **Remarks**

EndDoubleBuffer2D ends the double buffer mode and releases the memory allocated for the buffer.

### **Return Value**

On success, EndDoubleBuffer2D returns 0. On error, it returns a nonzero value.

### **See also**

[BeginDoubleBuffer2D](#), [UpdateBuffer2D](#)

## **UpdateBuffer3D**

### **Function**

Displays the buffered image in the double buffer mode.

### **Syntax**

short UpdateBuffer3D (HDC hdc, HVIEW hview);

### **Remarks**

UpdateBuffer3D 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, UpdateBuffer3D returns 0. On error, it returns a nonzero value.

### **See also**

[BeginDoubleBuffer3D](#), [EndDoubleBuffer3D](#)

## **UpdateBuffer2D**

### **Function**

Displays the buffered image in the double buffer mode.

### **Syntax**

short UpdateBuffer2D (HDC hdc, HVIEW hview);

### **Remarks**

UpdateBuffer3D 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, UpdateBuffer2D returns 0. On error, it returns a nonzero value.

### **See also**

[BeginDoubleBuffer2D](#), [EndDoubleBuffer2D](#)



## **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 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 a octahedron.

### Syntax

```
void Octahedron (HDC hdc, float r);
```

### Remarks

Octahedron draws a octahedron in the current 3D viewer with current pen color 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 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 a icosahedron.

### Syntax

```
void Icosahedron (HDC hdc, float r);
```

### Remarks

Icosahedron draws a icosahedron in the current 3D viewer with current pen color 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](#)

## **CreateViewer2D**

### **Function**

Creates a 2D viewer

### **Syntax**

HVIEW CreateViewer2D (NPSTR Name, int X, int Y, int Width, int Height);

### **Remarks**

CreateViewer2D creates a 2D viewer. The viewport dimension is Width by Height with Upper-left corner at (X ,Y). The name of the viewer is given by Name.

### **Return Value**

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

### **See also**

[InitialGraphics2D](#), [SetViewport2D](#)

## **CreateViewer3D**

### **Function**

Creates a 3D viewer.

### **Syntax**

HVIEW CreateViewer3D (NPSTR Name, int X, int Y, int Width, int Height);

### **Remarks**

CreateViewer3D creates a 3-D viewer. The viewport dimension is Width by Height with upper-left corner at (X , Y). The name of the viewer is given by Name.

### **Return Value**

The viewer handle will be returned if it is created successfully. Otherwise, NULL will be returned.

### **See also**

[InitialGraphics3D](#), [SetViewport3D](#)

## **Max2DViewer**

### **Function**

Sets the default maximum number of 2D viewers.

### **Syntax**

void Max2DViewer (short N);

### **Remarks**

Max2DViewer sets the default maximum number of 2D viewers to N.

### **Return value**

None.

### **See also**

[CreatViewer2D](#), [InitialGraphics2D](#)

## **Max3DViewer**

### **Function**

Sets the default maximum number of 3D viewers.

### **Syntax**

void Max3DViewer (short N);

### **Remarks**

Max3DViewer sets the default maximum number of 3D viewers as N.

### **Return vlaue**

None.

### **See also**

[CreatViewer3D](#), [InitialGraphics3Df\\_initialgraphics3d](#)



## **Max2DMStack**

### **Function**

Sets the default maximum depth of the 2D transformation matrix stack.

### **Syntax**

```
void Max2DMStack (short N);
```

### **Remarks**

Max2DMStack sets the default maximum depth of the 2D transformation matrix stack as N.

### **Return value**

None.

### **See also**

[InitialGraphics2D](#)

## **Max3DMStack**

### **Function**

Sets the default maximum depth of the 3D transformation matrix stack.

### **Syntax**

```
void Max3DMStack (short N);
```

### **Remarks**

Max3DMStack sets the default maximum depth of the 3D transformation matrix stack as N.

### **Return value**

None.

### **See also**

[InitialGraphics3D](#)

## **InitialGraphics2D**

### **Function**

Initializes the 2D graphic system.

### **Syntax**

short InitialGraphics2D (short nview, short npoint, short ndepth);

### **Remarks**

InitialGraphics2D initializes the 2D graphic system with the specified maximum number of viewers, points, and depth of matrix stack. If any of the numbers is set to zero, the default maximum number will be used.

### **Return value**

On successful completion, InitialGraphics2D returns 0. It returns a nonzero number on error.

### **See also**

[ExitGraphics](#), [Max2DViewer](#), [Max2DMStack](#)

## **InitialGraphics3D**

### **Function**

Initializes the 3D graphic system.

### **Syntax**

short InitialGraphics3D (short nview, short npoint, short ndepth);

### **Remarks**

InitialGraphics3D initializes the 3D graphic system with specifying the maximum number of viewers, points, and depth of matrix stack. If any number is set to zero, the default maximum number will be used.

### **See also**

[ExitGraphics](#), [Max2DViewer](#), [Max2DMStack](#)

## **ExitGraphics**

### **Function**

Exits the graphic system and free the memory used.

### **Syntax**

```
void ExitGraphics (void);
```

### **Remarks**

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

### **Return value**

None.

### **See Also**

[InitialGraphics2D](#), [InitialGraphics3D](#)

## **PenColor**

### **Function**

Selects 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](#)

## **PushMatrix2D**

### **Function**

Pushes the 2D transformation matrix stack.

### **Syntax**

short PushMatrix2D (void);

### **Remarks**

PushMatrix2D pushes the 2D transformation matrix stack. A copy of the stack top is pushed to the stack.

### **Return value**

PushMatrix2D returns 0 upon successful completion. A nonzero value is returned if the stack is full.

### **See also**

[PopMatrix2D](#)



## **PushMatrix3D**

### **Function**

Pushes the 3D transformation matrix stack.

### **Syntax**

short PushMatrix3D (void);

### **Remarks**

PushMatrix3D pushes the 3D transformation matrix stack. A copy of the stack top is pushed to the stack.

### **Return value**

On success, PopMatrix3D returns 0. A nonzero value is returned if the stack is full.

### **See also**

[PopMatrix3D](#)

## **PopMatrix2D**

### **Function**

Pops the 2D transformation matrix stack.

### **Syntax**

short PopMatrix2D (void);

### **Remarks**

PopMatrix2D pops the 2D transformation matrix stack. The stack top is discarded.

### **Return value**

On success, PopMatrix2D returns 0. A nonzero value is returned if the stack is empty.

### **See also**

[PushMatrix2D](#)

## **PopMatrix3D**

### **Function**

Pops the 3D transformation matrix stack.

### **Syntax**

short PopMatrix3D (void);

### **Remarks**

PopMatrix3D pops the 3D transformation matrix stack. The stack top is discarded.

### **Return value**

On success, PopMatrix3D returns 0. A nonzero value is returned if the stack is empty.

### **See also**

[PushMatrix3D](#)

## **SetView3D**

### **Function**

Sets 3D viewer's view transformation matrix.

### **Syntax**

short SetView3D (HVIEW Hview, float VX, float VY, float VZ, float RX, float RY, float RZ, float Twist);

### **Remarks**

SetView3D sets 3D viewer Hview's viewing transformation matrix according to the viewer position VX, VY, and VZ; the view reference RX, RY, and RZ; and the viewer Twist angle.

### **Return value**

On success, SetView3D returns 0. On error, it returns a nonzero value.

### **See also**

[SetPolarView3D](#)

## **SetPolarView3D**

### **Function**

Sets 3D viewer based on polar coordinates.

### **Syntax**

short SetPolarView3D (HVIEW Hview, float CX, float CY, float CZ, float Dist, float Azim, float Inc, float Twist);

### **Remarks**

SetPolarView3D sets viewer Hview's viewing transformation matrix according to the reference center CX, CY, and CZ; the Dist form the reference center to the viewer; and the three orientation angles Azim, Inc, and Twist.

### **Return value**

On success, SetPolarView3D returns 0. On error, it returns a nonzero value.

### **See also**

[SetView3D](#)

## **SetPerspective3D**

### **Function**

Sets perspective projection of a 3D viewer.

### **Syntax**

short SetPerspective3D (HVIEW Hview, float Fovy, float Aspect, float Front, float Back);

### **Remarks**

SetPerspective sets 3D viewer Hview's perspective projection matrix according to the field of view angle of Fovy, Aspect ratio, Front and Back clipping panes.

### **Return value**

On success, SetPerspective3D returns 0. On error, it returns a nonzero value.

### **See also**

[SetOrthogonal3D](#)

## **SetOrthogonal3D**

### **Function**

Sets orthogonal projection of a 3D viewer.

### **Syntax**

short SetOrthogonal3D (HVIEW Hview, float Left, float Right, float Bottom, float Top, float Front, float Back);

### **Remarks**

SetOrthogonal3D sets 3D viewer Hview's orthogonal projection matrix according to the viewing box defined by the Left, Right, Bottom, Top, Front, and Back .

### **Return value**

On success, SetOrthogonal3D returns 0. On error, it returns a nonzero value.

### **See also**

[SetPerspective3D](#)

## **SetViewport2D**

### **Function**

Sets a 2D viewer's viewport.

### **Syntax**

short SetViewport2D (HVIEW Hview, short X, short Y, short Width, short Height);

### **Remarks**

SetViewport2D sets 2D viewer Hview's viewport according to the upper left point (X,Y) and the Width and Height in display coordinates.

### **Return value**

On success, SetViewport2D returns 0. On error, it returns a nonzero value.

### **See also**

[GetViewport2D](#)



## **SetViewport3D**

### **Function**

Sets 3D viewer's viewport .

### **Syntax**

short SetViewport3D (HVIEW Hview, short X, short Y, short Width, short Height);

### **Remarks**

SetViewport3D sets 3D viewer Hview's viewport according to the upper left point (X,Y) and the Width and Height in display coordinates.

### **Return value**

On success, SetViewport3D returns 0. On error, it returns a nonzero value.

### **See also**

[GetViewport3D](#)

## **SetView2D**

### **Function**

Sets a 2D viewer's viewing transformation matrix .

### **Syntax**

short SetView2D (HVIEW Hview, float X, float Y, float Angle);

### **Remarks**

SetView2D sets 2D viewer's view transformation according to the center coordinates X, Y, and the rotation Angle.

### **Return value**

On success, SetView2D returns 0. On error, it returns a nonzero value.

### **See also**

[SetWindow2D](#)

## **SetProjection2D**

### **Function**

Sets 2D viewer's projection transformation.

### **Syntax**

short SetProjection2D (HVIEW 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 0. On error, it returns a nonzero value.

### **See also**

[SetWindow2D](#)

## **SetWindow2D**

### **Function**

Sets 2D viewer's viewing and projection transformations.

### **Syntax**

short SetWindow2D (HVIEW Hview, float X1, float Y1, float X2, float Y2);

### **Remarks**

Set 2D viewer's viewing transformation and projection transformation according to the two corner points in the world coordinates defined by X1, Y1, X2, and Y2.

### **Return value**

On success, SetWindow2D returns 0. On error, it returns a nonzero value.

### **See also**

[SetView2D](#), [SetProjection2D](#)

## SelectViewer3D

### Function

Selects a 3D viewer.

### Syntax

short SelectViewer3D (HVIEW hview);

### Remarks

SelectViewer3D selects viewer Hview as the current 3D viewer. The subsequent 3D drawing function calls will use this viewer. hview must be a valid viewer handle returned by CreateViewer3D.

### Return value

On success, SelectViewer3D returns 0. On error, it returns a nonzero value.

### See also

[CreateViewer3D](#)

## SelectViewer2D

### Function

Selects 2D viewer.

### Syntax

short SelectViewer2D (HVIEW hview);

### Remarks

SelectViewer2D selects viewer Hview as the current 2D viewer. The subsequent 2D drawing function calls will use this viewer. hview must be a valid viewer handle returned by CreateViewer2D.

### Return value

On success, SelectViewer2D returns 0. On error, it returns a nonzero value.

### See also

[CreateViewer2D](#)

## DisplayViewerFrame2D

### Function

Displays the frame of a 2D viewer.

### Syntax

short DisplayViewerFrame2D (HDC hdc, HVIEW hview, short color);

### Remarks

DisplayViewerFrame2D draws the 2D viewer Hview's rectangle border with Color. The frame is defined by the viewport set in the function CreateViewer2D or SetViewport2D.

### Return value

On success, DisplayViewerFrame2D returns 0. On error, it returns a nonzero value.

### See also

[CreateViewer2D](#), [SetViewport2D](#)

## **DisplayViewerFrame3D**

### **Function**

Displays the frame of a 3D viewer.

### **Syntax**

short DisplayViewerFrame3D (HDC hdc, HVIEW hview, short color);

### **Remarks**

DisplayViewerFrame3D draws the 3D viewer Hview's rectangle border with Color. The frame is defined by the viewport set in the function CreateViewer3D or SetViewport3D.

### **Return value**

On success, DisplayViewerFrame3D returns 0. On error, it returns a nonzero value.

### **See also**

[CreateViewer3D](#), [SetViewport3D](#)



## **DisplayViewerName2D**

### **Function**

Display 2D viewer's name.

### **Syntax**

short DisplayViewerName2D (HDC hdc, HVIEW hview, short color, short top);

### **Remarks**

DisplayViewerName2D displays the viewer Hview's name with Color.

### **Return value**

On success, DisplayViewerName2D returns 0. On error, it returns a nonzero value.

### **See also**

[GetViewerName2D](#), [SetViewerName2D](#)

## **DisplayViewerName3D**

### **Function**

Displays a 3D viewer's name.

### **Syntax**

short DisplayViewerName3D (HDC hdc, HVIEW hview, short color, short top);

### **Remarks**

Display 3D viewer Hview's name with Color.

### **Return value**

On success, DisplayViewerName3D returns 0. On error, it returns a nonzero value.

### **See also**

[GetViewerName3D](#), [SetViewerName3D](#)

## **ClearViewer2D**

### **Function**

Clears a 2D viewer.

### **Syntax**

short ClearViewer2D (HDC hDc, HVIEW hview, short color);

### **Remarks**

ClearViewer2D clears a 2D viewer Hview's client area with Color.

### **Return value**

On success, ClearViewer2D returns 0. On error, it returns a nonzero value.

### **See also**

[CreateViewer2D](#)

## **ClearViewer3D**

### **Function**

Clears a 3D viewer.

### **Syntax**

short ClearViewer3D (HDC hDc, HVIEW hview, short color);

### **Remarks**

ClearViewer3D clears a 3D viewer's client area with Color.

### **Return value**

On success, ClearViewer3D returns 0. On error, it returns a nonzero value.

### **See also**

[CreateViewer3D](#)

## **MoveViewer3D**

### **Function**

Moves a 3D viewer.

### **Syntax**

short MoveViewer3D (HVIEW Hview, float LeftRight, float UpDow, float BackForth);

### **Remarks**

MoveViewer3D moves the 3D viewer Hview in the view coordinate system according to LeftRight, UpDown, and BackForth.

### **Return value**

On success, MoveViewer3D returns 0. On error, it returns a nonzero value.

### **See also**

[MoveWorld3D](#)

## **MoveWorld3D**

### **Function**

Moves a 3D viewer.

### **Syntax**

short MoveWorld3D (HVIEW Hview, float X, float Y, float Z);

### **Remarks**

Moves 3D viewer Hview in the world coordinate system along X, Y, and Z.

### **Return value**

On success, MoveWorld3D returns 0. On error, it returns a nonzero value.

### **See also**

[MoveViewer3D](#)

## **RotateViewer3D**

### **Function**

Rotates a 3D viewer.

### **Syntax**

short RotateViewer3D (HVIEW Hview, float Yaw, float Pitch, float Twist);

### **Remarks**

RotateViewer3D rotates the 3D viewer Hview in the view coordinate system according to angles of Yaw, Pitch, and Twist with unit of degrees.

### **Return value**

On success, RotateViewer3D returns 0. On error, it returns a nonzero value.

### **See also**

[RotateWorld3D](#)

## **RotateWorld3D**

### **Function**

Rotates a 3D viewer.

### **Syntax**

short RotateWorld3D (HVIEW Hview, float X, float Y, float Z);

### **Remarks**

RotateWorld3D rotates the 3D viewer Hview in the world coordinate system around X, Y, and Z Axes with unit of degrees.

### **Return value**

On success, RotateWorld3D returns 0. On error, it returns a nonzero value.

### **See also**

[RotateViewer3D](#)



## **ZoomViewer3D**

### **Function**

Zooms a 3D viewer.

### **Syntax**

short ZoomViewer3D (HVIEW Hview, float Zoom);

### **Remarks**

ZoomViewer3D zooms the 3D viewer Hview by factor Zoom.

### **Return value**

On success, ZoomViewer3D returns 0. On error, it returns a nonzero value.

### **See also**

[ZoomWorld3D](#)

## **ZoomWorld3D**

### **Function**

Zooms a 3D viewer.

### **Syntax**

short ZoomViewer3D (HVIEW Hview, float Zoom);

### **Remarks**

ZoomViewer3D zooms the 3D viewer Hview by factor Zoom.

### **Return value**

On success, ZoomViewer3D returns 0. On error, it returns a nonzero value.

### **See also**

[ZoomViewer3D](#)

## **MoveViewer2D**

### **Function**

Moves a 2D viewer.

### **Syntax**

short MoveViewer2D (HVIEW hview, float LeftRight, float UpDown);

### **Remarks**

MoveViewer2D changes the viewing transformation of a 2D viewer Hview by moving it in the view coordinate system according to LeftRight and UpDown.

### **Return value**

On success, MoveViewer2D returns 0. On error, it returns a nonzero value.

### **See also**

[RotateViewer2D](#)

## **RotateViewer2D**

### **Function**

Rotates a 2D viewer.

### **Syntax**

short RotateViewer2D (HVIEW hview, float Angle);

### **Remarks**

RotateViewer2D rotates the 2D viewer Hview by Angle degrees.

### **Return value**

On success, RotateViewer2D returns 0. On error, it returns a nonzero value.

### **See also**

[MoveViewer2D](#)

## **Num2DViewer**

### **Function**

Gets the number of 2D viewers created in the current system.

### **Syntax**

short Num2DViewer (void);

### **Remarks**

Num2DViewer gets the number of 2D viewers created in the current system.

### **Return value**

Num2DViewer returns the number of 2D viewers.

### **See also**

[CreateViewer2D](#)

## **Num3DViewer**

### **Function**

Gets the number of 3D viewers created in the current system.

### **Syntax**

short Num3DViewer (void);

### **Remarks**

Num3DViewer gets the number of 3D viewers created in the current system.

### **Return value**

Num3DViewer returns the number of 3D viewers.

### **See also**

[CreatViewer3D](#)

## **ViewerPosition3D**

### **Function**

Gets a 3D viewer's position.

### **Syntax**

short ViewerPosition3D (HVIEW Hview, float \*VX, float \*VY, float \*VZ);

### **Remarks**

ViewerPosition3D gets the 3D viewer Hview's position in the world coordinate system VX, VY, and VZ .

### **Return value**

On success, ViewerPosition3D returns 0. On error, it returns a nonzero value.

### **See also**

[SetView3D](#)

## **ViewerOrientation3D**

### **Function**

Gets a 3D viewer's orientation.

### **Syntax**

short ViewerOrientation3D (HVIEW Hview, float \*Azim, float \*Inc, float \*Twist);

### **Remarks**

ViewerOrientation3D gets 3D viewer Hview's orientation in the world coordinate system Azim, Inc, and Twist.

### **Return value**

On success, ViewerOrientation3D returns 0. On error, it returns a nonzero value.

### **See also**

[SetPolarView3D](#)



## **ViewerDirection3D**

### **Function**

Gets a 3D viewer's direction.

### **Syntax**

short ViewerDirection3D (HVIEW Hview, float \*X, float \*Y, float \*Z);

### **Remarks**

ViewerDirection3D gets the 3D viewer Hview's direction vector's three components X, Y, and Z on the three axes of the world coordinate system .

### **Return value**

On success, ViewerDirection3D returns 0. On error, it returns a nonzero value.

### **See also**

[SetView3D](#)

## **ViewProjectionMode3D**

### **Function**

Get 3D viewer Hview's projection mode.

### **Syntax**

short ViewProjectionMode3D (HVIEW Hview);

### **Remarks**

### **Return value**

### **See also**

## **ViewerPosition2D**

### **Function**

Gets a 2D viewer's center position.

### **Syntax**

short ViewerPosition2D (HVIEW Hview, float \*CX, float \*CY, float \*Angle);

### **Remarks**

ViewerPosition2D gets the 2D viewer Hview's center position CX, CY and the rotation Angle in the world coordinate system.

### **Return value**

On success, ViewerPosition2D returns 0. On error, it returns a nonzero value.

### **See also**

[SetView2D](#)

## ViewerField3D

### Function

Gets a 3D viewer's viewing field.

### Syntax

```
short ViewerField3D (HVIEW Hview, float *Left, float *Right, float *Bottom, float *Top,  
float *Width, float *Height);
```

### Remarks

ViewerField3D gets the 3D viewer Hview's viewing field defined by Left, Right, Bottom, Top, Front, and Back in the view coordinate system.

### Return value

On success, ViewerField3D returns 0. On error, it returns a nonzero value.

### See also

[SetPerspective3D](#), [SetOrthogonal3D](#)

## **ViewerField2D**

### **Function**

Gets a 2D viewer's viewing field.

### **Syntax**

short ViewerField2D (HVIEW 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 rotates on the current 3D transformation matrix (the stack top) by the amount specified. Axis can be 'x', 'y', or 'z'. Angle is measured in degrees.

### **Return value**

None

### **See also**

[Translate3D](#), [Scale3D](#)

## **Translate3D**

### **Function**

Translates on the current 3D transformation matrix.

### **Syntax**

```
void Translate3D (float X, float Y, float Z);
```

### **Remarks**

Translate3D performs a 3D modeling transformation on the current 3D transformation matrix by a translation of (X,Y,Z).

### **Return value**

None.

### **See also**

[Rotate3D](#), [Scale3D](#)

## Scale3D

### Function

Scales on the current 3D transformation matrix .

### Syntax

```
void Scale3D (float X, float Y, float Z);
```

### Remarks

Scale3D scales on the current 3D transformation matrix (the stack top) in the x , y, and z directions by the amounts specified.

### Return value

None.

### See also

[Translate3D](#), [Rotate3D](#)



## **Translate2D**

### **Function**

Translates on the current 2D transformation matrix.

### **Syntax**

```
void Translate2D (float X, float Y);
```

### **Remarks**

Translate2D translates on the current 2D transformation matrix (the stack top) in the x and y directions by the amounts specified.

### **Return value**

None.

### **See also**

[Rotate2D](#), [Scale2D](#)

## **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 amounts specified.

### **Return value**

None.

### **See also**

[Translate2D](#), [Scale2D](#)

## Scale2D

### Function

Scales on the current 2D transformation matrix.

### Syntax

```
void Scale2D (float x, float y);
```

### Remarks

Scale2D scales on the current 2D transformation matrix (the stack top) in the x and y directions by the amounts specified.

### Return value

None.

### See also

[Translate2D](#), [Rotate2D](#)

## **GetViewerName2D**

### **Function**

Gets the name of a 2D viewer.

### **Syntax**

short GetViewerName2D (HVIEW hview, NPSTR name);

### **Remarks**

GetViewerName2D gets the name string of the 2D viewer hview.

### **Return value**

On success, GetViewerName2D returns 0. On error, it returns a nonzero value.

### **See also**

[DisplayViewerName2D](#), [SetViewerName2D](#)

## **SetViewerName2D**

### **Function**

Sets the name of a 2D viewer.

### **Syntax**

short SetViewerName2D (HVIEW hview, NPSTR name);

### **Remarks**

SetViewerName2D sets the name string of the 2D viewer hview.

### **Return value**

On success, SetViewerName2D returns 0. On error, it returns a nonzero value.

### **See also**

[DisplayViewerName2D](#), [GetViewerName2D](#)

## **GetViewerName3D**

### **Function**

Gets the name of a 3D viewer.

### **Syntax**

short GetViewerName3D (HVIEW hview, NPSTR name);

### **Remarks**

GetViewerName3D gets the name string of the 3D viewer hview.

### **Return value**

On success, GetViewerName3D returns 0. On error, it returns a nonzero value.

### **See also**

[DisplayViewerName3D](#), [SetViewerName3D](#)

## **SetViewerName3D**

### **Function**

Sets the name of a 3D viewer.

### **Syntax**

short SetViewerName3D (HVIEW hview, NPSTR name);

### **Remarks**

SetViewerName3D sets the name string of the 3D viewer hview.

### **Return value**

On success, SetViewerName3D returns 0. On error, it returns a nonzero value.

### **See also**

[DisplayViewerName3D](#), [GetViewerName3D](#)

## **GetViewport2D**

### **Function**

Gets the position of a 2D viewport.

### **Syntax**

short GetViewport2D (HVIEW Hview, short \*X, short \*Y, short \*Width, short \*Height);

### **Remarks**

GetViewport2D gets the 2D viewer Hview's viewport position in display coordinates as the upper-left corner X and Y, and the Width and Height.

### **Return value**

On success, GetViewport2D returns 0. On error, it returns a nonzero value.

### **See also**

[SetViewport2D](#)



## **GetViewport3D**

### **Function**

Gets the position of a 3D viewport.

### **Syntax**

short GetViewport3D (HVIEW Hview, short \*X, short \*Y, short \*Width, short \*Height);

### **Remarks**

GetViewport3D gets 3D viewer Hview's viewport position in display coordinates as the upper-left corner X and Y, and the Width and Height.

### **Return value**

On success, GetViewport3D returns 0. On error, it returns a nonzero value.

### **See also**

[SetViewport3D](#)

## **MoveTo2D**

### **Function**

Moves to a new position.

### **Syntax**

```
void MoveTo2D (HDC hDC, float X, float Y);
```

### **Remarks**

MoveTo2D moves the current 2D display position to X and Y in the current viewer.

### **Return value**

None.

### **See also**

[LineTo2D](#)

## **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 and Y in the current viewer with the current pen.

### **Return value**

None.

### **See also**

[MoveTo2D](#)

## **DrawLine2D**

### **Function**

Draws a 2D line segment.

### **Syntax**

```
void DrawLine2D (HDC hDC, float X1, float Y1, float X2, float Y2);
```

### **Remarks**

DrawLine2D draws a 2D line from X1 and Y1 to X2 and Y2 in the current 2D viewer with the current pen.

### **Return value**

None.

### **See also**

[LineTo2D](#), [MoveTo2D](#)

## **Polyline2D**

### **Function**

Draws a 2D polyline.

### **Syntax**

void Polyline2D (HDC hdc, LPPOINT2D Point, short N);

### **Remarks**

Polyline2D draws a 2D polyline defined by N 2D Points in the current 2D viewer with current pen for edges and current brush for interior.

### **Return value**

None.

### **See also**

[Polygon2D](#)

## **Polygon2D**

### **Function**

Draws a 2D polygon.

### **Syntax**

void Polygon2D (HDC hdc, LPPOINT2D point, short count);

### **Remarks**

Polygon2D draws a 2D polygon defined by N 2D Points in the current 2D viewer with current pen for edges and current brush for interior.

### **Return value**

None.

### **See also**

[Polyline2D](#)

## **ResetVertex2D**

### **Function**

Reset temporary 2D point buffer Point2D to empty.

### **Syntax**

```
void ResetVertex2D (void);
```

### **Remarks**

### **Return value**

None.

### **See also**

## **SetVertex2D**

### **Function**

Set one 2D point X and Y into the 2D point buffer Point2D

### **Syntax**

```
void SetVertex2D (float X, float Y);
```

### **Remarks**

### **Return value**

None.

### **See also**



## **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 X1, Y1, X2, and Y2 in the current 2D viewer with current pen for edge and current brush for interior.

### **Return value**

None.

### **See also**

[Polygon2D](#)

## Circle2D

### Function

Draws a 2D circle..

### Syntax

```
void Circle2D (HDC hdc, float X, float Y, float Radius);
```

### Remarks

Circle2D draws a 2D circle defined by center X, Y and Radius in the current 2D viewer with current pen for edge and current brush for interior.

### Return value

None.

### See also

[Arc2D](#)

## **Arc2D**

### **Function**

Draws a 2D circular arc.

### **Syntax**

void Arc2D (HDC hdc, float x, float y, float r, float start, float angle);

### **Remarks**

Arc2D draws a 2D circular arc in the current 2D viewer with the current pen color. (x,y) is the center of the circle and r is the radius of the circle. The starting angle and the span of the arc are specified by the parameter start and angle measured in degrees.

### **Return Value**

None.

### **See also**

[Circle2D](#)

## **EllipseArc2D**

### **Function**

Draws a 2D elliptic arc.

### **Syntax**

void EllipseArc2D (HDC hDC, float x, float y, float r1, float r2, float start, float angle);

### **Remarks**

EllipseArc2D draws a 2D elliptic arc in the current 2D viewer with the current pen color. (x,y) is the center of the ellipse and r1, r2 are the half-axes of the ellipse. The starting angle and the span of the arc are specified by the parameter start and angle measured in degrees.

### **Return Value**

None.

### **See also**

[Ellipse2D](#)

## Wedge2D

### Function

Draws a 2D circular wedge.

### Syntax

```
void Wedge2D (HDC hdc, float x, float y, float r1, float r2, float start, float angle);
```

### Remarks

Wedge2D draws a 2D circular wedge in the current 2D viewer with the current pen color. (x,y) is the center of the circle and r is the radius of the circle. The starting angle and the span of the arc are specified by the parameter start and angle measured in degrees.

### Return Value

None.

### See also

[Arc2D](#)

## **Ellipse2D**

### **Function**

Draws a 2D ellipse.

### **Syntax**

```
void Ellipse2D (HDC hDC, float x, float y, float r1, float r2);
```

### **Remarks**

Ellipse2D draws a 2D ellipse defined by center X, Y and two radius R1 and R2 in the current 2D viewer with current pen for edge and current brush for interior.

### **Return value**

None.

### **See also**

[EllipseArc2D](#)

## **Ngon2D**

### **Function**

Draws a 2D N-sided regular polygon.

### **Syntax**

```
void Ngon2D (HDC hDC, float X, float Y, float R1, float R2, short N);
```

### **Remarks**

Ngon2D draws a 2D N-sided regular polygon define by ceter X, Y and two radius R1 and R2 in the current 2D viewer with current pen for edge and current brush for interior.

### **Return value**

None.

### **See also**

[Polygon2D](#)

## **NsideStar2D**

### **Function**

Draws a 2D N-point star.

### **Syntax**

void NsideStar2D (HDC hDC, float X, float Y, float Radius, short N);

### **Remarks**

NsideStar2D draws a 2D N-point regular star defined by center X, Y and Radius in the current 2D viewer with current pen for edge and current brush for interior.

### **Return value**

None.

### **See also**

[Polygon2D](#)



## **NsideFlower2D**

### **Function**

Draw a 2D N-point flower.

### **Syntax**

```
void NsideFlower2D (HDC hDC, float X, float Y, float R1, float R2, short N);
```

### **Remarks**

Draw a 2D N-point flower defined by center X, Y, and the inner and outer radius R1 and R2 in the current 2D viewer with current pen for edge and current brush for interior

### **Return value**

None.

### **See also**

[Polygon2D](#)

## **MoveTo3D**

### **Function**

Move 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, and Z in the current 3D viewer.

### **Return value**

None.

### **See also**

[LineTo3D](#)

## **LineTo3D**

### **Function**

Draws a 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, and Z in the current viewer with current pen.

### **Return value**

None.

### **See also**

[MovoTo3D](#)

## **DrawLine3D**

### **Function**

Draws a 3D line segment.

### **Syntax**

```
void DrawLine3D (HDC hDC, float X1, float Y1, float Z1, float X2, float Y2, float Z2);
```

### **Remarks**

DrawLine3D draws a 3D line from X1, Y1, and Z1 to X2, Y2, and Z2 in the current 3D viewer with current pen.

### **Return value**

None.

### **See also**

[LineTo3D](#), [MovoTo3D](#)

## **MarkPosition3D**

### **Function**

Draws 3D axes.

### **Syntax**

void MarkPosition3D (HDC hdc, float x, float y, float z, float scale);

### **Remarks**

MarkPosition3D draws a 3D axes in size of Scale at X, Y, and Z in the current 3D viewer with red, green, and blue for the three axes.

### **Return value**

None.

### **See also**

[CreateViewer3D](#)

## **Polyline3D**

### **Function**

Draw a 3D polyline.

### **Syntax**

void Polyline3D (HDC hdc, LPPOINT3D Point, short N);

### **Remarks**

Polyline3D draws a polyline defined by N 3D Point with current pen for the edge and current brush for the interior in the current viewer.

### **Return value**

None.

### **See also**

[Polygon3D](#)

## **Polygon3D**

### **Function**

Draws a 3D polygon.

### **Syntax**

void Polygon3D (HDC hdc, LPPOINT3D Point, short N);

### **Remarks**

Polygon3D draws a polygon defined by N 3D Point with current pen for the edge and current brush for the interior in the current viewer.

### **Return value**

None.

### **See also**

[Polyline3D](#)

## **ResetVertex3D**

Function

Reset the temporary 3D point buffer Point3D to empty

Syntax

```
void ResetVertex3D (void);
```

**Remarks**

**Return value**

**See also**



## **SetVertex3D**

### Function

Set a 3D point X, Y, and Z into the 3D point buffer Point3D

### Syntax

```
void SetVertex3D (float X, float Y, float Z);
```

### **Remarks**

#### **Return value**

None.

### **See also**

## **Shape3D**

### **Function**

Draw a 2D shape defined by N 2D Point at X, Y, and Z in MS Windows' device context hDC with current pen for the edge and current brush for the interior

### **Syntax**

void Shape3D (HDC hDC, float X, float Y, float Z, LPPOINT2D Point, short N);

### **Remarks**

Draw a 2D shape defined by N 2D Point at X, Y, and Z in MS Windows' device context hDC with current pen for the edge and current brush for the interior

### **Return value**

None.

### **See also**

## **Rectangle3D**

### **Function**

Draw a Rectangle.

### **Syntax**

void Rectangle3D (HDC hDC, float X1, float Y1, float X2, float Y2, float Z);

### **Remarks**

Rectangle3D draws a Rectangle defined by X1, Y1, X2, and Y2 and elevation Z in the current 3D viewer with current pen for the edge and current brush for the interior.

### **Return value**

None.

### **See also**

[Polygon3D](#)

## **Prism3D**

### **Function**

Draws a 3D prism.

### **Syntax**

```
void Prism3D (HDC hDC, float X, float Y, float Z, float H, LPPOINT2D BaseVertex,  
LPPOINT2D HeadVertex, short N);
```

### **Remarks**

Prism3D draws a 3D prism defined by the N point base shape BaseVertex and head shape HeadVertex at X, Y, and Z with current pen for the facets edges and the current brush for the facet interior.

### **Return value**

None.

### **See also**

[NsidePrism3D](#)

## **Pyramid3D**

### **Function**

Draws a pyramid.

### **Syntax**

void Pyramid3D (HDC hDC, float x, float y, float z, float height, LPPOINT2D basevertex, short count);

### **Remarks**

Pyramid3D draws a pyramid in the current 3D viewer. The apex is specified by (x,y,z). The vertices of the based is in the array basevertex and the number of base vertices is given by count.

### **Return Value**

None.

### **See also**

[NsidePyramid3D](#)

## **NsideStar3D**

### **Function**

Draw a 3D N point star.

### **Syntax**

```
void NsideStar3D (HDC hDC, float X, float Y, float Z, float H, float R, short N);
```

### **Remarks**

NsideStar3D draw a 3D N point regular star defined by H and R and X, Y, and Z with current pen for the facet edges and the current brush for the facet interior.

### **Return value**

None.

### **See also**

[NsideFlower3D](#)

## **NsideFlower3D**

### **Function**

Draw a 3D N point flower.

### **Syntax**

```
void NsideFlower3D (HDC hDC, float X, float Y, float Z, float H, float R1, float R2, short N);
```

### **Remarks**

NsideFlower3D draws a 3D N point regular flower defined by R1, R2, and H at X, Y, and Z with current pen for the facet edges and the current brush for the facet interior.

### **Return value**

None.

### **See also**

[NsideStar3D](#)

## **Cube3D**

### **Function**

Draws a 3D rectangular box.

### **Syntax**

```
void Cube3D (HDC hDC, float x1, float y1, float z1, float x2, float y2, float z2);
```

### **Remarks**

Cube3D draws a 3D rectangular box defined by two corner points X1, Y1, Z1 and X2, Y2, Z2 with current pen for the facet edges and the current brush for the facet interior

### **Return value**

None.

### **See also**

[Rectangle3D](#)



## **Sphere3D**

### **Function**

Draws a sphere.

### **Syntax**

void Sphere3D (HDC hDC, float X, float Y, float Z, float R, short N1, short N2);

### **Remarks**

Sphere3D draws a spherical polyhedron with radius R at X, Y, and Z in the current 3D viewer with current pen for the facet edges and the current brush for the facet interior

### **Return value**

None.

### **See also**

[Cylinder3d](#), [Cone3D](#)

## **NsidePyramid3D**

### **Function**

Draws a regular 3D pyramid.

### **Syntax**

```
void NsidePyramid3D (HDC hDC, float X, float Y, float Z, float R, float H, short N);
```

### **Remarks**

Draw a vertical 3D N sided regular pyramid defined by radius R and height H at position X, Y, and Z in the current viewer with current pen for the facet edges and the current brush for the facet interior

### **Return value**

None.

### **See also**

[Pyramid3D](#)

## **Cone3D**

### **Function**

Draws a cone.

### **Syntax**

```
void Cone3D (HDC hDC, float X, float Y, float Z, float R, float H);
```

### **Remarks**

Cone3D draws a vertical 3D cone define by radius R and height H at position X, Y, and Z in the current viewer with current pen for the facet edges and the current brush for the facet interior

### **Return value**

None.

### **See also**

[Cylinder3D](#)

## **NsidePrism3D**

### **Function**

Draws an N sided regular prism.

### **Syntax**

```
void NsidePrism3D (HDC hDC, float X, float Y, float Z, float R, float H, short N);
```

### **Remarks**

Draw a vertical 3D N side prism defined by radius R and height H at X, Y, and Z with current pen for the facet edges and the current brush for the facet interior.

### **Return value**

None.

### **See also**

[Prism3D](#)

## Cylinder3D

### Function

Draw a 3D cylinder.

### Syntax

```
void Cylinder3D (HDC hdc, float X, float Y, float Z, float R, float H);
```

### Remarks

Cylinder3D draws a vertical 3D cylinder defined by radius R and height H at position X, Y, and Z in the current 3D viewer with current pen for the facet edges and the current brush for the facet interior.

### Return value

None.

### See also

[Cone3D](#)

## **BezierCurve2D**

### **Function**

Draws a 2D Bezier curve.

### **Syntax**

```
void BezierCurve2D(HDC hdc, LPPOINT2D CtrlPolygon);
```

### **Remarks**

BezierCurve2D draws a Bezier curve in the current 2D viewer. The curve is specified by four control points in the CtrlPolygon.

### **Return value**

None.

### **See also**

[BSplineCurve2D](#), [HermitCurve2D](#), [NURBSCurve2D](#)

## **HermitCurve2D**

### **Function**

Draws a 2D Hermit curve.

### **Syntax**

```
void HermitCurve2D(LPPOINT2D CtrlPolygon);
```

### **Remarks**

HermitCurve2D draws a Hermit curve in the current 2D viewer. The curve is specified by four control points in the CtrlPolygon.

### **Return value**

None.

### **See also**

[BezierCurve2D](#), [BSplineCurve2D](#), [NURBSCurve2D](#)

## **BSplineCurve2D**

### **Function**

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

### **Syntax**

```
void BSplineCurve2D(HDC hdc, LPPOINT2D CtrlPolygon, int N);
```

### **Remarks**

BezierCurve2D draws a uniform non-rational B-Spline curve in the current 2D viewer. The curve is specified by  $N$  control points in the CtrlPolygon. The first and the last knots are of multiplicity 3 and all other 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, LPPOINT2D CtrlPolygon, int N, float 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 in the CtrlPolygon and  $N+2$  Knots.

### **Return value**

None.

### **See also**

[BezierCurve2D](#), [BSplineCurve2D](#), [HermitCurve2D](#)

## **BezierCurve3D**

### **Function**

Draws a 3D Bezier curve.

### **Syntax**

```
void BezierCurve3D(HDC hdc, LPPOINT3D CtrlPolygon);
```

### **Remarks**

BezierCurve3D draws a Bezier curve in the current 3D viewer. The curve is specified by four control points in the CtrlPolygon.

### **Return value**

None.

### **See also**

[BSplineCurve3D](#), [HermitCurve3D](#), [NURBSCurve3D](#)

## **HermitCurve3D**

### **Function**

Draws a 3D Hermit curve.

### **Syntax**

```
void HermitCurve3D(LPPOINT3D CtrlPolygon);
```

### **Remarks**

HermitCurve3D draws a Hermit curve in the current 3D viewer. The curve is specified by four control points in the CtrlPolygon.

### **Return value**

None.

### **See also**

[BezierCurve3D](#), [BSplineCurve3D](#), [NURBSCurve3D](#)

## **BSplineCurve3D**

### **Function**

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

### **Syntax**

```
void BSplineCurve3D(HDC hdc, LPPOINT3D CtrlPolygon, int N);
```

### **Remarks**

BezierCurve3D draws a uniform non-rational B-Spline curve in the current 3D viewer. The curve is specified by  $N$  control points in the CtrlPolygon. The first and the last knots are of multiplicity 3 and all other 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, LPPOINT3D CtrlPolygon, int N, float 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 in the CtrlPolygon and  $N+2$  Knots.

### **Return value**

None.

### **See also**

[BezierCurve3D](#), [BSplineCurve2D](#), [HermitCurve3D](#)

## **BezierSurface3D**

### **Function**

Draws a 3D Bezier surface.

### **Syntax**

```
void BezierSurface3D(HDC hdc, LPPOINT3D CtrlNet, int Ns, int Nt);
```

### **Remarks**

BezierSurface3D draws a Bezier surface in the current 3D viewer. The curve is specified by the CtrlNet which is an array of 4 by 4 points. The surface is drawn in wire-frame form with  $Ns+1$  lines in s direction and  $Nt+1$  lines in the t direction.

### **Return value**

None.

### **See also**

[BSplineSurface3D](#), [HermitSurface3D](#), [NURBSSurface3D](#)

## **HermitSurface3D**

### **Function**

Draws a 3D Hermit surface.

### **Syntax**

```
void HermitSurface3D(LPPOINT3D CtrlNet);
```

### **Remarks**

HermitCurve3D draws a Hermit curve in the current 3D viewer. The curve is specified by four control points in the CtrlPolygon. The surface is drawn in wire-frame form with  $N_s+1$  lines in s direction and  $N_t+1$  lines in the t direction.

### **Return value**

None.

### **See also**

[BezierSurface3D](#), [BSplineSurface3D](#), [NURBSSurface3D](#)

## **BSplineSurface3D**

### **Function**

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

### **Syntax**

```
void BSplineSurface3D(HDC hdc, LPPOINT3D CtrlPolygon, int N, int Ns, int Nt);
```

### **Remarks**

BSplineSurface3D draws a uniform non-rational B-Spline surface in the current 3D viewer. The curve is specified by  $N$  control points in the CtrlPolygon. The first and the last knots are of multiplicity 3 and all other knots are simple and uniformly spaced. The surface is drawn in wire-frame form with  $N_s+1$  lines in  $s$  direction and  $N_t+1$  lines in the  $t$  direction for each rectangular patch.

### **Return value**

None.

### **See also**

[BezierCurve3D](#), [HermitCurve3D](#), [NURBSCurve3D](#)



## **NURBSSurface3D**

### **Function**

Draws a 3D NURBS surface.

### **Syntax**

```
void NURBSSurface3D(HDC hdc, LPPOINT3D CtrlPolygon, float SKnots[], float TKnot[],  
int SCount, int TCount, int Ns, int Nt);
```

### **Remarks**

NURBSSurface3D draws a non-uniform rational B-spline (NURBS) surface in the current 3D viewer. The surface is specified by SCount by TCount control points in the CtrlPolygon and with SCount+2 SKnots and TCount+2 TKnots. The surface is drawn in wire-frame form with Ns+1 lines in s direction and Nt+1 lines in the t direction for each bezier patch.

### **Return value**

None.

### **See also**

[BezierSurface3D](#), [BSplineSurface2D](#), [HermitSurface3D](#)

## Appendix A. BIBLIOGRAPHY

Gerald Farin,

Alan Watt,

## Appendix B. COMMON QUESTIONS

Q. What is the difference between your MoveTo2D, LineTo2D and GDI's MoveTo, LineTo functions?

A. MoveTo2D and LineTo2D performs the transformation from the world coordinates to the viewport of the selected viewer. MoveTo and LineTo uses the screen coordinates. GDI has several screen mapping modes, but they are all simple scaling transformations. Visualib provides much more sophisticated viewing transformations. Another difference is that GDI functions use 16 bit integer type for coordinates which may easily cause overflow, while Visualib functions use float type.

Q. I just want to display some simple 2D graphics. How could Visualib help me?

A. GDI drawing functions have several limitations. For example, GDI Ellipse function can only draw ellipses with horizontal and vertical axes. Visualib lets you draw any kinds of ellipses with its powerful transformation capabilities.