View3D Overview

This help file documents the View3D Delphi Component.

The View3D component allows an application to display three dimensional data (in the form of polygons) on a **form (TForm)**.

The data to be displayed consists of a number of models (see TView3DModel). Each model consists of a number of polygons. It is these polygons that define the shape of the data.

The models to be displayed are added to a view (see TView3D) by creating a reference object (see TView3DReference) that refers to the model. The reference object can then be added to the view's list of objects to be displayed.

Reference objects can have a position and orientation, which places them in the appropriate place in the view. This position and orientation may be calculated by the user or by use of a TView3DRoute object.

The view has a number of parameters that control how the models will appear when they are displayed. Such parameters include lighting, fog, shading and filling.

View3D uses a left-handed co-ordinate system. That is the positive x-axis is to the right, the position y-axis is upwards and the positive z-axis is into the screen. This is equivalent to x being the easting, z the northing and y the height.

All View3D objects inherit a constructor called Create and a destructor called Destroy. These procedures should be called to create and destroy a View3D object.

Code Sample

Type TView3D

Unit
view3d

Description
TView3D is an installable Delphi component.

TView3D allows for the display of 3D polygon data on a **form (TForm)**.

TView3D encapsulates all the parameters required to view 3D objects. These parameters include the lighting conditions, how the objects should be drawn and the position of the eye.

TView3D has the following properties that can be assigned via the Object Inspector.

The following are properties of TView3D:

```
property Ambient    : TView3DColor;
property Background : TView3DColor;
property Cull       : Boolean;
property Eye        : TView3DPosition;
property Fill       : TView3DFill:
property Fog        : TView3DFog;
property Lights     : TView3DLights;
property Shade      : TView3DShade;
property Size       : TView3DSize;
property Volume     : TView3DVolume;
```

TView3D has the following methods:

```
procedure AddReference(reference : PView3DReference);
procedure Draw;
procedure RemoveReference(reference : PView3DReference);
function  GetBitmap : TBitmap;
```

Property TView3D.Ambient

TView3D.Ambient is the ambient light color in the view.

```
property Ambient : TView3DColor;
```

Description
TView3D.Ambient defines the ambient light color in the view. It specifies the red, green and blue color components.

The ambient light color is the light which illuminates polygons that are not being directly illuminated by any other light source. It can be thought of as the background lighting.

The property has read and write access. The default value is (0.0, 0.0, 0.0).

Property TView3D.Background

TView3D.Background is the background color of the view.

```
property Background : TView3DColor;
```

Description
TView3D.Background defines the background color of the view. It specifies the red, green and blue color components.

Background does not affect the appearance of any object.

The property has read and write access. The default value is (0.0, 0.0, 0.0).

Property TView3D.Cull

TView3D.Cull is the state of back face culling.

```
property Cull : Boolean;
```

Description
TView3D.Cull specifies the state of back face culling. When an object is drawn, those polygons which are facing away from the eye point need not always be drawn and may be quickly discarded. This can improve the drawing speed.

When TView3D.Cull is True, back facing polygons will be removed and not displayed. When TView3D.Cull is False, all polygons will be considered for drawing.

The property has read and write access. The default value is False.

Property TView3D.Eye

TView3D.Eye is the position of the eye.

```
property Eye : TView3DPosition;
```

Description
TView3D.Eye specifies the position and orientation of the eye. This is the position at which the eye can be thought of as being positioned. The view drawn will be as that seen from this position.

The property has read and write access. The default value is at the origin.

Property TView3D.Fill

TView3D.Fill is the fill style of the polygons.

```
property Fill : TView3DFill;
```

Description
TView3D.Fill specifies how each polygon that is drawn should be filled.

The property has read and write access. The default value is Solid.

Property TView3D.Fog

TView3D.Fog is the fog within the view.

```
property Fog : TView3DFog;
```

Description
TView3D.Fog specifies the state of the fog within the view.

The property has read and write access. The default value is the fog is disabled (TView3DFog.State equals False).

Property TView3D.Lights

TView3D.Lights defines the eight light sources in the view.

```
property Lights : TView3DLights;
```

Description
TView3D.Lights is an array of eight lights sources (see TView3DLights), which controls the lighting within the view.

The property has read and write access. The default value for each light is off (TView3DLight.State equals False).

Property TView3D.Shade

TView3D.Shade is the method used for shading the polygons.

```
property Shade : TView3DShade;
```

Description
TView3D.Shade specifies the shading to be performed on each polygon. If Shade is Flat then the color of each polygon will be constant across the whole of it's surface. If Shade is Smooth, the color will vary across the polygon's surface.

The property has read and write access. The default value is Smooth.

Property TView3D.Size

TView3D.Size is the size and position of the view.

```
property Size : TView3DSize;
```

Description
TView3D.Size specifies the area of the view within the form.

The property has read and write access.

The default size is 1 x 1.

Property TView3D.Volume

TView3D.Volume is the projection within the view.

```
property Volume : TView3DVolume;
```

Description
TView3D.Volume specifies the perspective projection within the view.

The property has read and write access.

The default value of TView3DVolume.Angle is 45.0, the default value of TView3DVolume.Back is 10000.0, the default value of TView3DVolume.Front is 1.0, the default value of TView3DVolume.Scale is 1.0, and default value of TView3DVolume.Projection is Perspective.

Procedure TView3D.AddReference

TView3D.AddReference adds a reference (see TView3DReference) to the view.

```
procedure AddReference(reference : PView3DReference);
```

Description
TView3D.AddReference adds a reference to the view. If the reference is already in the view, this method does nothing. To add multiple copies of a model (see TView3DModel) to a view a reference should be created for each copy required.

Procedure TView3D.RemoveReference

TView3D.RemoveReference removes a reference (see TView3DReference) from the view.

```
procedure RemoveReference(reference : PView3DReference);
```

Description
TView3D.RemoveReference removes a reference from the view. If the reference is not in the view, this method does nothing.

Procedure TView3D.Draw

TView3D.Draw draws the view.

```
procedure Draw;
```

Description
TView3D.Draw draws (displays) the view. Each reference *(*see TView3DReference*)* is displayed in it's current position taking into account the currently lighting, fog effects, shading and fill modes *(*see TView3D*)*.

Function TView3D.GetBitmap

TView3D.GetBitmap returns the view as a bitmap.

```
function GetBitmap : Tbitmap;
```

Description
TView3D.GetBitmap creates a bitmap (by calling TBitmap.Create) and copies the current view display contents to it. If an error occurs nil is returned, otherwise the bitmap is returned.

Once the application has finished using the returned bitmap, it should be destroyed.

Type TView3DColor

TView3DColor is the class that defines a color in terms of the red green and blue components.

Unit
mat3d

Description
TView3DColor is the View3D object that defines a color.

A color consists of three components : red, green and blue.
Each of these components is a Byte value.

The following are properties of TView3DColor :

```
property Red   : Byte;
property Green : Byte;
property Blue  : Byte;
```

These properties have read and write access. The default value for each is zero.

Type TView3DPosition

TView3DPosition is the class that defines a position and orientation.

Unit
eye3d

Description
TView3DPosition is the View3D object that defines a position and orientation. This may be for the eye point (see TView3D)or for a reference (see TView3DReference).

The following are properties of TView3DPosition :

```
property X     : Single;
property Y     : Single;
property Z     : Single;
property Pitch : Single;
property Yaw   : Single;
property Roll  : Single;
```

View3D uses a left-handed co-ordinate system. That is the positive x-axis is to the right, the position y-axis is upwards and the positive z-axis is into the screen. This is equivalent to x being the easting, z the northing and y the height.

TView3DPosition.X is the distance along the x-axis, TView3DPosition.Y is the distance along the y-axis and TView3DPosition.Z is the distance along the z-axis. The units of TView3DPosition.X, TView3DPosition.Y and TView3DPosition.Z can be whatever the application requires.

TView3DPosition.Pitch is the angle of rotation about the x-axis, TView3DPosition.Yaw the angle of rotation about the y-axis and TView3DPosition.Roll the angle of rotation about the z-axis. These angles are measured in degrees.

These properties have read and write access. The default value for each is zero.

Type TView3DFill

TView3DFill is the View3D set of fill modes.

Unit
view3d

Description
TView3DFill is the set of fill modes that can be applied to a view (see TView3D).

```
TView3DFill = ( Point, Line, Solid );
```

The fill mode will be used to draw the polygons that are in the view.

When the fill mode is Point only the polygon vertices will be drawn.

When the fill mode is Line only the polygon edges will be drawn.

When the fill mode is Solid the polygon will be filled.

Type TView3DShade

TView3DShade is the View3D set of shade modes.

Unit
view3d

Description
TView3DShade is the set of shade modes that can be applied to a view (see TView3D).

```
TView3DShade = ( Flat, Smooth );
```

The shade mode will be used to draw the polygons that are in the view.

When the shade mode is Flat the color of a polygon is determined by the color of it's first vertex. This color is dependent on the material of the polygon (see TView3DMaterial), and the lighting in the view (see TView3DLights). This can produce images of objects that appear faceted. This may be overcome by firstly smoothing the model (see TView3DModel) and then using the shade mode set to Smooth. If fog is enabled (see TView3DFog), then the color at each vertex will be calculated based on the polygon color, but also taking into account it's distance with respect to the fog. This may lead to a variation of color over the surface of the polygon.

When the shade mode is Smooth, the color at each vertex is calculated taking into account the material of the polygon and the lighting in the view. Fog will be applied as for the case when the shade mode is Flat.

Type TView3DFog

TView3DFog is the View3D object that defines fog.

Unit
light3d

Description
TView3DFog is the View3D object that defines the state of the fog within the view.

TView3DFog has the following properties:

```
property Back    : Single;
property Color   : TView3DColur;
property Density : Single;
property Front   : Single;
property Mode    : TView3DFogMode;
property State   : Boolean;
```

If fog is enabled in a view, then fog blends a color, the fog color(TView3DFog Color) (see TView3DColor), with the color of a pixel from the polygon. The pixel color is dependent on the shade mode (see TView3DShade). Fog is enabled by setting TView3DFog.State to True.

The blending factor is calculated in one of three ways depending on the value of TView3DFog.Mode (see TView3DFogMode).

When TView3DFog.Mode has the value Linear, the factor (f) is such that if Z is the distance of the pixel then:

if z <= TView3DFog.Front then f := 0.0
else if z >= TView3DFog.Back then f := 1.0
else f := (z - TView3DFog.Front) / (TView3DFog.Back - TView3DFog.Front);

Thus TView3DFog.Front and TView3DFog.Back define the range of distances over which the fog acts.

When TView3DFog.Mode has the value Exp then the factor (f) has the value:
f := Exp(-TView3DFog.Density * Z);

and when TView3DFog.Mode has the value Exp2 then the factor (f) has the value:
f := Exp(-TView3DFog.Density * Z * Z);

Thus TView3DFog.Density defines the fog density.

It is usual (although not compulsory) for the value of TView3DFog.Color to be the same as the background color (see TView3D.Background).

The properties have read and write access. The default values for TView3DFog.Back, TView3DFog.Front and TView3DFog.Density are zero. The default value of TView3DFog.Color is (0.0, 0.0, 0.0). The default value of TView3DFog.State is False and for TView3DFog.Mode is Linear.

Type TView3DFogMode

TView3DFogMode is the View3D set of fog modes.

Unit
light3d

Description
TView3DFogMode is the View3D set of fog modes.

```
TView3DFogMode = ( Linear, Exp, Exp2 );
```

Type TView3DSize

TView3DSize is the View3D object that defines the view size.

Unit
view3d

Description
TView3DSize defines the position and size of the view within the form.

TView3DSize has the following properties:

```
property Bottom : Cardinal;
property Height : Cardinal;
property Left   : Cardinal;
property Width  : Cardinal;
```

TView3DSize.Bottom specifies the position of the bottom edge of the view in number of lines. This is measured from the bottom of the form. Thus, a value of 10 would mean the view is 10 lines up from the bottom of the form.

TView3D.Height specifies the height of the view in lines.

TView3D.Left specifies the position of the left edge of the view in number of pixels. This is measured from the left edge of the form.

TView3D.Width specifies the width of the view in pixels.

The properties have read and write access. The default values for all properties is zero.

Type TView3DVolume

TView3DVolume is the View3D object that defines the view projection.

Unit
eye3d

Description
TView3DVolume defines how the contents of a view are projected.

TView3DVolume has the following properties:

```
property Angle      : Single;
property Back       : Single;
property Front      : Single;
property Projection : TView3DProjection;
property Scale      : Single;
```

TView3DVolume.Angle defines the horizontal field of *view* and is in degrees.

TView3DVolume.Back defines a back clipping plane. Polygons that are beyond this distance and hence are on the opposite side of the back clipping plane to the eye point are not drawn. Those polygons that intersect the back clipping plane will be clipped and only those fragments which lie on the same side of the clipping plane as the eye point will be drawn.

TView3DVolume.Front defines a front clipping plane. Polygons that are closer than this distance and hence are on the same side of the front clipping plane as the eye point are not drawn. Those polygons that intersect the front clipping plane will be clipped and only those fragments which lie on the opposite side of the clipping plane from the eye point will be drawn.

TView3DVolume.Projection defines the type of projection to use in the view. If TView3DVolume.Projection is equal to Perspective then TView3DVolume.Angle defines the angular field of view. If TView3DVolume.Projection is equal to Orthographic then TView3DVolume.Scale defines the scale factor for the view.

TView3DVolume.Scale defines the horizontal scaling of the *view*.

The units of TView3DVolume.Back and TView3DVolume.Front are those of the view.

When fog is enabled (see TView3DFog), and the fog color is the same as the background color (see TView3D.Background), the value of TView3DVolume.Back is usually set to the value of TView3DFog.Back. This is an optimisation that prevents polygons that would be drawn in the same color as the background, and hence invisible, from being drawn.

The properties have read and write access. The default values for all properties is zero.

Type TView3DProjection

TView3DProjection is the View3D set of projection modes.

Unit
eye3d

Description
TView3DProjection is the set of projection modes that can be applied to a view volume (see TView3DVolume).

```
TView3DProjection = ( Perspective, Orthographic );
```

A mode that is Perspective will produce a perspective view, this is the ‘normal’ representation of a three-dimensional scene. A mode that is Orthographic will produce a orthographic or parallel view. Such views can be useful for displaying engineering type views or for plan view displays.

Type TView3DLights

TView3DLights is the View3D object that defines the lights in a view.

Unit
light3d

Description
TView3DLights is a record of eight TView3DLight objects.

TView3DLights has the following properties:

```
property Light0                  : TView3DLight;
property Light1                  : TView3DLight;
property Light2                  : TView3DLight;
property Light3                  : TView3DLight;
property Light4                  : TView3DLight;
property Light5                  : TView3DLight;
property Light6                  : TView3DLight;
property Light7                  : TView3DLight;
property Light[Index : Integer] : TView3DLight;
```

Thus, for example, light 2 may be address either by using Light2 or Light[2].

Each light may individually be controlled.

Type TView3DLight

TView3DLight is the View3D object that defines a light source.

Unit
light3d

Description
TView3DLight defines a single light. There are eight light sources in a view (see TView3D).

TView3DLight has the following properties:

```
property Azimuth   : Single;
property Color     : TView3DColor;
property Elevation : Single;
property State     : Boolean;
```

A light may be regarded as an infinite light source much like the Sun. It's position is given by the two angles TView3DLight.Azimuth and TView3DLight.Elevation. These angles are in degrees.

The color of the light is given TView3DLight.Color.

A light may be disabled by setting TView3DLight.State to False and enabled by setting TView3DLight.State to True.

The properties have read and write access. The default values for TView3DLight.Azimuth and TView3DLight.Elevation are zero. The default value of TView3DLight.Color is (0.0, 0.0, 0.0). The default value of TView3DLight.State is False;

Type TView3DMaterial

TView3DMaterial is the View3D object that defines the appearance of a polygon.

Unit
mat3d

Description
TView3DMaterial defines the visual appearance of a polygon.

TView3DMaterial has the following properties:

```
property Ambient      : TView3DColor;
property Diffuse      : TView3DColor;
property Specular     : TView3DColor;
property Shininess    : Single;
property Emission     : TView3DColor;
property Transparency : Single;
property Texture      : PView3DTexture;
```

Each polygon in a model (see TView3DModel) has as one of it's properties a material that is of the type TView3DMaterial.

TView3DMaterial.Ambient defines the color of the material due to ambient lighting.

TView3DMaterial.Diffuse defines the color of the material due to diffuse lighting. It is this color that is generally the perceived color of a material.

TView3DMaterial.Specular defines the color of the material due to any specular highlights. TView3DMaterial.Shininess controls the size of this highlight.

TView3DMaterial.Emission is a color that always contributes to the polygon color irrespective of the lighting conditions.

TView3DMaterial.Transparency is a number in the range [0.0, 1.0] which controls the transparency of the polygon. A value of 0.0 indicates the polygon is fully transparent (i.e. invisible). A value of 1.0 indicates the polygon is fully opaque.

TView3D.Texture is a pointer to a View3D texture object (see TView3Dtexture). If this property is not nil then a texture will be applied to the polygon. Not that this property is a pointer. This allows multiple materials to use the same texture.

Generally TView3DMaterial.Ambient and TView3DMaterial.Diffuse will be the same color but with a different brightness. For example the ambient value may be (127, 0, 0) and the diffuse value (255, 0, 0).

The properties have read and write access. The default values for TView3DMaterial.Ambient is (127, 127, 127). The default value for TView3DMaterial.Diffuse is (255, 255, 255). The default value for TView3DMaterial.Specular is (0, 0, 0). The default values for TView3DMaterial.Shininess and TView3DMaterial.Transparency is 0.0. The default value TView3DMaterial.Emission is (0.0, 0.0, 0.0). The default value for TView3DMaterial.Texture is nil.

Type TView3DTexture

TView3DTexture is the View3D object that defines a texture.

Unit
text3d

Description
TView3DTexture is the View3D object that defines a texture map that can be used with a material (see TView3DMaterial) to affect the appearance of a polygon.

TView3DTexture has the following properties:

```
property Bitmap  : TBitmap;
property Combine : TView3DTextureCombine;
property Quality : TView3DTextureQuality;
```

When written to TView3DTexture.Bitmap, the bitmap is converted to an internal format that is suitable for use by assigning the address of the TView3DTexure object to a material. The original bitmap is still suitable for use by the application.

TView3DTexture.Combine defines how the texture should be applied. If the value of TView3DTexture.Combine is Decal then the color of the polygon, derived from the material, will be ignored and the texture will be applied directly. If the value is Modulate, then the color of the texture map will be combined with that of the polygon. This can be used to generate lit textured scenes.

TView3DTexture.Quality defines the quality of the texture map. If the value of TView3DTexture.Quality is Low only a single point sample of the texture map will be taken at each displayed pixel in a polygon. If the value is High then up to four samples will be taken and filtered (bilinear filtering).

Type TView3DTextureCombine

TView3DTextureCombine is the View3D set of texture combine modes.

Unit
text3d

Description
TView3DTextureCombine is the View3D set of texture combine modes.

```
TView3DTextureCombine = ( Modulate, Decal );
```

Type TView3DTextureQuality

TView3DTextureQuality is the View3D set of texture quality modes.

Unit
text3d

Description
TView3DTextureQuality is the View3D set of texture quality modes.

```
TView3DTextureQuality = ( Low, High);
```

Type TView3DModel

TView3DModel is the View3D object that is a collection of polygons.

Unit
model3d

Description
The data to be displayed in a view is constructed from polygons. Polygons need to be gathered together into localised models. The View3D object for these models is TView3DModel.

TView3DModel has the following methods:

```
procedure Empty;
function  AddPolygon(v1 : TView3DVertex;
                     v2 : TView3DVertex;
                     v3 : TView3DVertex;
                     var mat : TView3DMaterial) : Boolean;

function  AddLine(v1 : TView3DVertex;
                  v2 : TView3DVertex;
                  var mat : TView3DMaterial) : Boolean;

procedure Smooth(angle : Single; distance : Single);
```

Individual polygons can be added to a model by calling TView3DModel.AddPolygon where the parameters are the vertices of the polygon and it's material.

Individual lines can be added to a model by calling TView3DModel.AddLine where the parameters are the vertices of the line and it's material.

Either of these functions will return True if memory cannot be allocated to store the object.

All of the polygons and lines in a model may be removed by calling TView3DModel.Empty.

To remove the faceted appearance of a polygonal model the method TView3DModel.Smooth can be called. This will attempt to calculate sufficient information to remove or greatly reduce the distinct polygons edges.

The value of the parameter angle should be the largest angle (in degrees) between two polygons, that have a common edge, where the common edge should be made to be indistinct. A typical value for this parameter is 60.0.

The value of the parameter distance is the maximum distance between two polygon vertices that can be regarded as representing the same point. A typical value for this parameter might be 0.01.

Type TView3DVertex

TView3DVertex is the View3D record that represents a vertex.

Unit
model3d

Description
Each polygon consists of a number of vertices. Each of these vertices is a TView3DVertex record.

TView3DVertex has the following fields:

```
point   : TView3DPoint3;
texture : TView3DPoint2;
```

TView3DVertex.point is the co-ordinate of the vertex. That is the x, y, z position of the vertex relative to some application specific origin.

TView3DVertex.texture is the texture co-ordinate at the vertex. That is, TView3DVertex.texture specifies the x, y position in the texture map that should be at this vertex.

Type TView3DPoint3

TView3DPoint3 is the View3D record that represents a three dimensional point.

Unit
math3d

Description
TView3DPoint3 is a record that defines a point in three dimensions relative to some application specific origin.

TView3DPoint3 has the following fields:
```
x : Single;
y : Single;
z : Single;
```

Type TView3DPoint2

TView3DPoint2 is the View3D record that represents a two dimensional point.

Unit
math3d

Description
TView3DPoint2 is a record that defines a point in two dimensions relative to some application specific origin.

TView3DPoint2 has the following fields:
```
x : Single;
y : Single;
```

Type TView3DReference

TView3DReference is the View3D object that represents a reference to a model.

Unit
ref3d

Description
To be able to view a model (see TView3DModel) in the view (see TView3D) a reference needs to be created. TView3DReference is a link between the model and the view.

TView3DReference has the following properties:
```
property Model    : PView3DModel;
property Position : TView3DPosition;
```

TView3DReference.Model is a pointer to a TView3DModel object. This defines the model that this reference refers to. The default value is nil.

TView3DReference.Position defines the position of the reference with respect to some application origin. This allows the model to be moved without modifying the actual polygon data. The default value is zero for all fields. Use can be made of a TView3DRoute object to calculate a position.

Both these properties have read and write access.

TView3DReference has the following methods:
```
procedure Add(reference : PView3DReference);
procedure Remove(reference : PView3DReference);
```

TView3DReference.Add adds one reference to another. In this way object hierarchies can be constructed. Each reference may have a list of sub-references contained within it.

TView3DReference.Remove removes a reference from another.

A reference is added to the view by calling TView3D.AddReference.

Type TView3DRouteWayPt

TView3DRouteWayPt is the View3D record that represents a way point on a route.

Unit
route3d

Description
Each TView3DRoute object is made up of a list of way points. These way points define a position, orientation and speed. These way points are a TView3DRouteWayPt record.

TView3DRouteWayPt has the following fields:

```
x     : Single;
y     : Single;
z     : Single;
pitch : Single;
yaw   : Single;
roll  : Single;
speed : Single;
```

x is the distance along the x-axis, y is the distance along the y-axis and z is the distance along the z-axis. The units of x, y and z can be whatever the application requires.

pitch is the angle of rotation about the x-axis, yaw the angle of rotation about the y-axis and roll the angle of rotation about the z-axis. These angles are measured in degrees.

speed is the speed of movement at the way point. This has units of length per unit time, where length is the unit of x, y and z.

Type TView3DRoute

TView3DRoute is the View3D object that represents a route.

Unit
route3d

Description
A TView3DRoute object is a list of way points. Each way point defines the position, orientation and speed an object will have at that point. Each way point is a TView3DRouteWayPt record. The time an object gets to a particular way point is calculated internally, based upon the speeds at the way points. Not that the speed at a way point should never be zero, unless it is the last way point in the route.

Each way point has an index specify where it is in the list of way points.

TView3DRoute has the following methods:
```
procedure Add(waypt : TView3DRouteWayPt);
procedure Clear;
procedure Evaluate(time : Single);
procedure Insert(index : Cardinal; waypt : TView3DRouteWayPt);
procedure Remove(index : Cardinal);
function  Time(index : Cardinal) : Single;
```

TView3DRoute has the following properties:
```
property Position   : TView3DPosition;
property Continuous : Boolean;
property Count      : Cardinal;
property Period     : Single;
property TimeBase   : Single;
```

TView3DRoute.Add adds a way point to the route at the end of the list.
TView3DRoute.Clear removes all way points from the route.
TView3DRoute.Evaluate evaluates the route at the given time value. The value of TView3DRoute.Position then contains the calculated position and may be assigned to a TView3DReference object position.

TView3DRoute.Insert inserts the given way point before the way point with the given index.

TView3DRoute.Remove removes the way point with the given index.

TView3DRoute.Time returns the calculated time that an object will arrive at the way point with the give index.

The property TView3DRoute.Continuous specifies whether of not the route should continually loop. If the value is TRUE then the route will continually loop. This property has read and write access and is initialised to be FALSE.

The property TView3DRoute.Count specifies how many way points there are in the route. This property has read access only.

The property TView3DRoute.Period specifies the period of the route, that is the time it takes to get to the last node. This property has read access only.

The property TView3DRoute.TimeBase specifies am amount of time to shift the route in, that is the time is incremented by this value which may be negative. This property has read and write access and is initialised to zero.

Code Sample

The following is an example of the use of the View3D component. The source code and project is in the sample directory.

```
{ ========================================================== }

unit Unit1;

interface

uses
   Windows,
   Messages,
   SysUtils,
   Classes,
   Graphics,
   Controls,
   Forms,
   Dialogs,
   ExtCtrls,
   View3D,
   Mat3D,
   Model3D,
   Math3D,
   Ref3D,
   Route3D,
   Text3D;

{ ========================================================== }

const
   CylinderRadius : Single   = 10.0;
   CylinderHeight : Single   = 20.0;

{ ========================================================== }

type
   TForm1 = class(TForm)
      View3D1 : TView3D;
      Timer1  : TTimer;
      procedure FormCreate(Sender: TObject);
      procedure FormMouseMove(Sender: TObject;
                              Shift: TShiftState; X, Y: Integer);
      procedure FormDestroy(Sender: TObject);
      procedure FormResize(Sender: TObject);
      procedure FormPaint(Sender: TObject);
      procedure Timer1Timer(Sender: TObject);

      private
         procedure ConstructCylinder;
         procedure ConstructRoute;
         procedure Draw;
         procedure LoadTextures;

      public
```

```
         child1   : TView3DReference;
         child2   : TView3DReference;
         material : array [0..11] of TView3DMaterial;
         model    : TView3DModel;
         parent   : TView3DReference;
         radius   : Single;
         texture  : array[0..5] of TView3DTexture;
         route    : TView3DRoute;
         time     : Single;
   end;

{ ========================================================== }

var
   Form1 : TForm1;

{ ========================================================== }

implementation

{$R *.DFM}

{ ========================================================== }
{ === Procedure to draw the view                        === }
{ ========================================================== }
procedure TForm1.Draw;

begin
   View3D1.Draw;
end;

{ ========================================================== }
{ === Procedure to construct the view, called when the === }
{ === form is created                                   === }
{ ========================================================== }
procedure TForm1.FormCreate(Sender: TObject);

begin
   { === create a model === }
   model := TView3DModel.Create;

   { === create some references === }
   parent := TView3DReference.Create;
   child1 := TView3DReference.Create;
   child2 := TView3DReference.Create;

   { === set the child references to reference the model === }
   child1.Model := @model;
   child2.Model := @model;

   { === offset the position of the two child references === }
   child1.Position.X := CylinderRadius;
   child2.Position.X := -CylinderRadius;

   { === add the child references to the parent === }
   parent.Add(@child1);
   parent.Add(@child2);
```

```
   { === load textures === }
   LoadTextures();

   { === build a cylinder === }
   ConstructCylinder();

   { === calculate polygon vertex normals === }
   model.Smooth(-180.0, 0.0001);

   { === add reference to view === }
   View3D1.AddReference(@parent);

   { === initialise eye radius === }
   radius := Sqrt(View3D1.Eye.X * View3D1.Eye.X +
                  View3D1.Eye.Y * View3D1.Eye.Y +
                  View3D1.Eye.Z * View3D1.Eye.Z);

   { === construct route === }
   ConstructRoute();

end;

{ ========================================================= }
{ === Procedure to load textures                        === }
{ ========================================================= }
procedure TForm1.LoadTextures();

var
   bitmap : TBitmap;
   i      : Integer;
   name   : TFileName;

begin
   { === initialise texture bitmap file name === }
   name := 'textureX.bmp';

   { === create a bitmap === }
   bitmap := TBitmap.Create;

   { === load six textures === }
   for i := 0 to 5 do begin
      name[8] := Char(i + Integer('1'));

      { === create a texture === }
      texture[i] := TView3DTexture.Create;

      { === load bitmap from file === }
      bitmap.LoadFromFile(name);

      { === assign bitmap to texture === }
      texture[i].Bitmap  := bitmap;

      { === set texture quality and combination mode === }
      texture[i].Quality := Low;
      texture[i].Combine := Modulate;
   end;
```

```
end;

{ ========================================================= }
{ === Procedure to construct a 12-sided cylinder       === }
{ ========================================================= }
procedure TForm1.ConstructCylinder();

var
   angle  : Single;
   i      : Integer;
   pt     : array [0..23] of TView3DPoint3;
   vertex : array [0..3] of TView3DVertex;

begin
   { === setup points for === }
   for i := 0 to 11 do begin
      angle := i * 2.0 * Pi / 12;
      pt[i].x := CylinderRadius * Sin(angle);
      pt[i].y := -CylinderHeight;
      pt[i].z := CylinderRadius * Cos(angle);
      pt[i + 12].x := pt[i].x;
      pt[i + 12].y := CylinderHeight;
      pt[i + 12].z := pt[i].z;
   end;

   { === setup polygons === }
   for i := 0 to 11 do begin

      { === create a material === }
      material[i] := TView3DMaterial.Create;

      { === set material color and texture === }
      material[i].Diffuse.Red   := 255;
      material[i].Diffuse.Green := 255;
      material[i].Diffuse.Blue  := 255;
      material[i].texture := @texture[i mod 6];

      { === setup vertices === }
      vertex[0].point := pt[i];
      vertex[0].texture.x := 0.0;
      vertex[0].texture.y := 0.0;

      vertex[1].point := pt[(i + 1) mod 12];
      vertex[1].texture.x := 1.0;
      vertex[1].texture.y := 0.0;

      vertex[2].point := pt[(i + 1) mod 12 + 12];
      vertex[2].texture.x := 1.0;
      vertex[2].texture.y := 1.0;

      vertex[3].point := pt[i + 12];
      vertex[3].texture.x := 0.0;
      vertex[3].texture.y := 1.0;

      { === add polygons (two triangles) to model === }
      model.AddPolygon(vertex[0], vertex[1], vertex[2], @material[i]);
      model.AddPolygon(vertex[0], vertex[2], vertex[3], @material[i]);
```

```
   end;
end;

{ ======================================================== }
{ === Procedure to construct a simple route            === }
{ ======================================================== }
procedure TForm1.ConstructRoute();

var
   w : array[0..3] of TView3DRouteWayPt;

begin
   { === create route object === }
   route := TView3DRoute.Create;

   { === initialise way points === }
   w[0].x := -50.0;
   w[0].y := 0.0;
   w[0].z := 0.0;
   w[0].pitch := 0.0;
   w[0].yaw   := 0.0;
   w[0].roll  := 0.0;
   w[0].speed := 25.0;

   w[1].x := 0.0;
   w[1].y := 200.0;
   w[1].z := radius;
   w[1].pitch := 0.0;
   w[1].yaw   := 90.0;
   w[1].roll  := 0.0;
   w[1].speed := 25.0;

   w[2].x := 50.0;
   w[2].y := 0.0;
   w[2].z := 0.0;
   w[2].pitch := 0.0;
   w[2].yaw   := 180.0;
   w[2].roll  := 0.0;
   w[2].speed := 25.0;

   w[3].x := 0.0;
   w[3].y := 0.0;
   w[3].z := -radius * 1.5;
   w[3].pitch := 0.0;
   w[3].yaw   := 270.0;
   w[3].roll  := 0.0;
   w[3].speed := 25.0;

   { === add way points === }
   route.Add(w[0]);
   route.Add(w[1]);
   route.Add(w[2]);
   route.Add(w[3]);

   { === set route to auto-loop === }
   route.continuous := True;
```

```
   { === initialise time === }
   time := 0.0;
end;

{ ======================================================== }
{ === Procedure to change the eye point depending on   === }
{ === the mouse position                               === }
{ ======================================================== }
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y:
Integer);

var
   lat  : Single;
   long : Single;

begin

   { === calculate yaw value === }
   View3D1.Eye.Yaw := (X - ClientWidth / 2) / ClientWidth * 360.0;

   { === calculate pitch value === }
   View3D1.Eye.Pitch := (Y - ClientHeight / 2) / ClientHeight * 180.0;

   { === calculate x, y and z values === }
   long := DegToRad(View3D1.Eye.Yaw) + Pi;
   lat := DegToRad(View3D1.Eye.Pitch);
   View3D1.Eye.X := radius * Sin(long) * Cos(lat);
   View3D1.Eye.Y := radius * Sin(lat);
   View3D1.Eye.Z := radius * Cos(long) * Cos(lat);
end;

{ ======================================================== }
{ === Procedure to destroy objects                     === }
{ ======================================================== }
procedure TForm1.FormDestroy(Sender: TObject);

var
   i : Integer;

begin
   { === empty model === }
   model.Empty;

   { === destroy model === }
   model.Destroy;

   { === destroy references === }
   parent.Destroy;
   child1.Destroy;
   child2.Destroy;

   { === destroy textures === }
   for i := 0 to 5 do begin
      texture[i].Destroy;
   end;

   { === destroy materials === }
```

```
   for i := 0 to 11 do begin
      material[i].Destroy;
   end;

   { === destroy route === }
   route.Destroy;

end;

{ ========================================================= }
{ === Procedure to redraw the view when the form is    === }
{ === resized                                          === }
{ ========================================================= }
procedure TForm1.FormResize(Sender: TObject);

begin
   { === set new width & height === }
   View3D1.Size.Width  := ClientWidth;
   View3D1.Size.Height := ClientHeight;

   { === draw === }
   Draw;
end;

{ ========================================================= }
{ === Procedure to redraw the view when the form is    === }
{ === repainted                                        === }
{ ========================================================= }
procedure TForm1.FormPaint(Sender: TObject);

begin
   { === draw === }
   Draw;
end;

{ ========================================================= }
{ === Procedure to handle timer for animation          === }
{ ========================================================= }
procedure TForm1.Timer1Timer(Sender: TObject);

begin
   { === rotate references === }
   child1.Position.Pitch := child1.Position.Pitch + 10.0;
   child2.Position.Yaw   := child2.Position.Yaw - 10.0;

   { === evaluate route === }
   route.Evaluate(time);

   { === inrement time === }
   time := time + 0.1;

   { === set parent position === }
   parent.Position := route.Position;

   { === draw === }
   Draw();
end;
```

```
end.

{ ========================================================== }
```