14

Customizing 3D Analyst

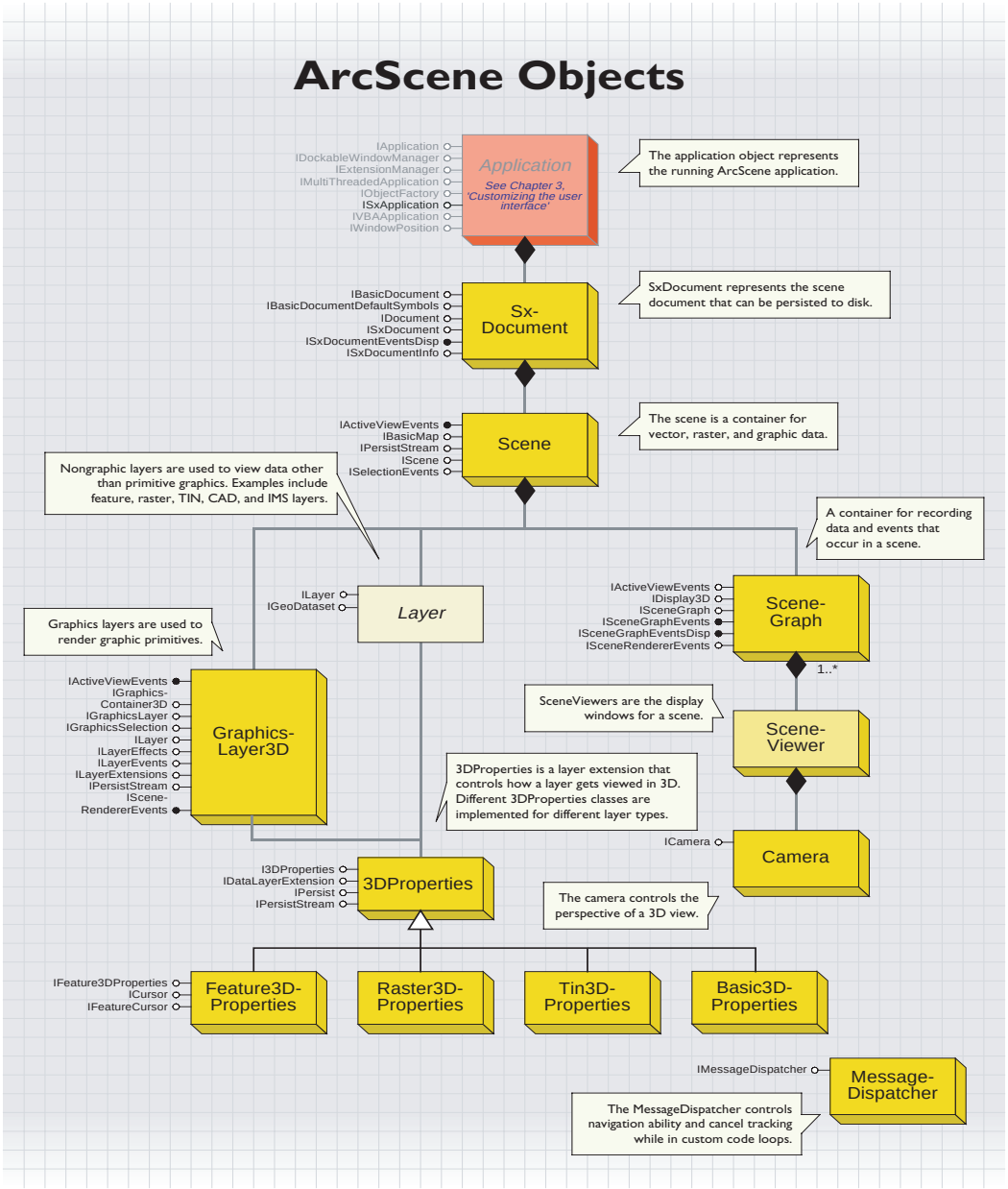
Clayton Crawford

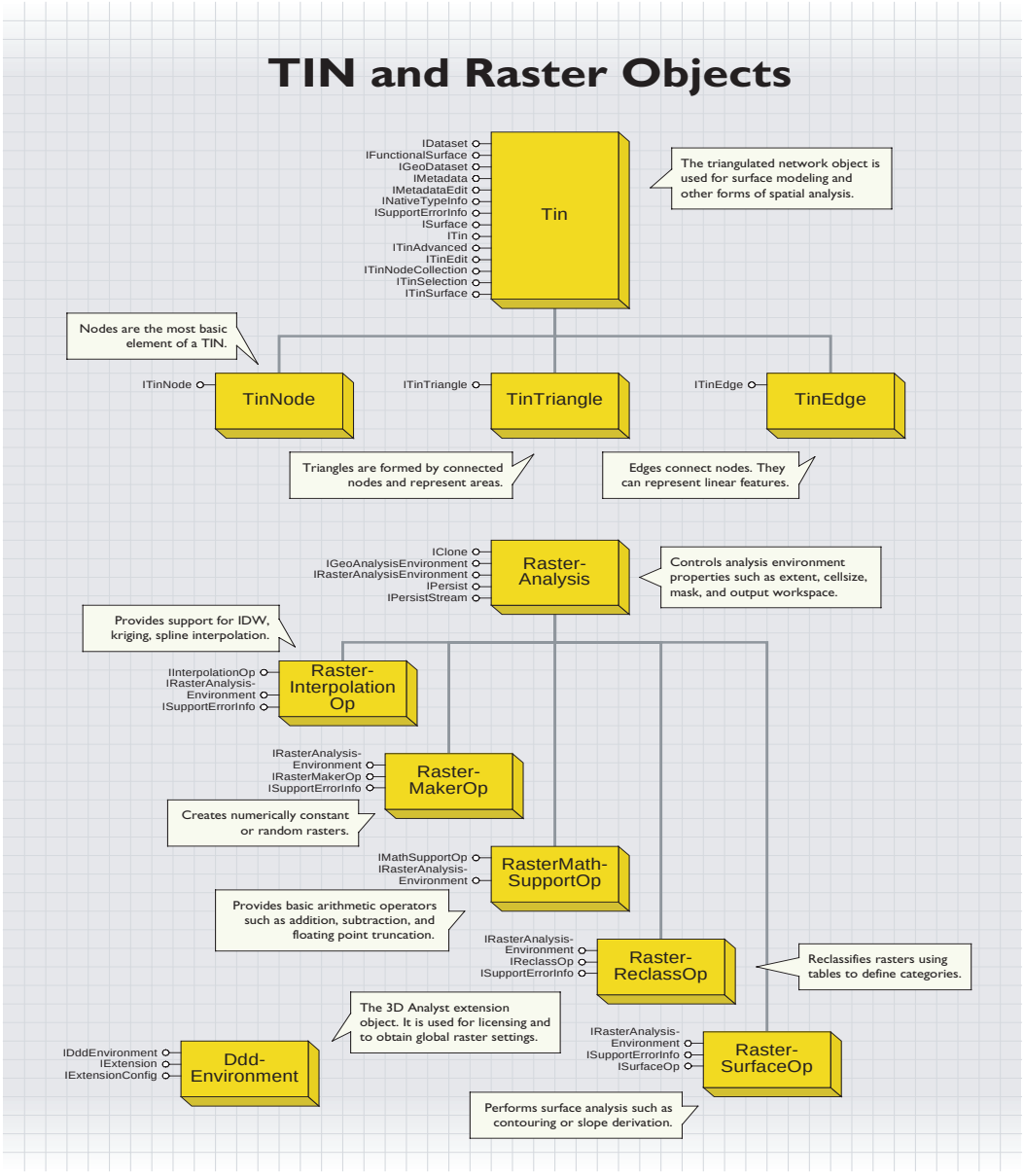
The purpose of this chapter is to provide a foundation for using the ESRI® ArcGIS™ 3D Analyst™ objects and their interfaces. Conceptual explanation coupled with sample code and graphics is used as a framework.

Visual Basic® (VB) code is used throughout this chapter for examples. This is because the built-in customization environment for ArcMap™, ArcCatalog™, and ArcScene™ is Visual Basic for Applications (VBA). If you're creating your own Dynamic Link Libraries (DLLs), any language that supports COM can be substituted. If you intend on using a different language, the authors hope the concepts are communicated effectively without being clouded by VB-specific syntax.

This text assumes you are already familiar with the basics of ArcObjects™ programming. For more information on the subject, please refer to the book Exploring ArcObjects.

For documentation on customizing the ArcScene user interface, refer to the customization chapter in Using ArcMap. *The same concepts apply for both applications.*


ArcScene Objects
IApplication
IDockableWindowManager
IExtensionManager
IMultiThreadedApplication
IObjectFactory
ISxApplication
IVBAApplication
IWindowPosition
Application
See Chapter 3, 'Customizing the user interface'
The application object represents the running ArcScene application.
IBasicDocument
IBasicDocumentDefaultSymbols
IDocument
ISxDocument
ISxDocumentEventsDisp
ISxDocumentInfo
Sx-Document
SxDocument represents the scene document that can be persisted to disk.
IActiveViewEvents
IBasicMap
IPersistStream
IScene
ISelectionEvents
Scene
The scene is a container for vector, raster, and graphic data.
Nongraphic layers are used to view data other than primitive graphics. Examples include feature, raster, TIN, CAD, and IMS layers.
A container for recording data and events that occur in a scene.
ILayer
IGeoDataset
Layer
Graphics layers are used to render graphic primitives.
IActiveViewEvents
IDisplay3D
ISceneGraph
ISceneGraphEvents
ISceneGraphEventsDisp
ISceneRendererEvents
Scene-Graph
1..*
IActiveViewEvents
IGraphics-Container3D
IGraphicsLayer
IGraphicsSelection
ILayer
ILayerEffects
ILayerEvents
ILayerExtensions
IPersistStream
IScene-RendererEvents
Graphics-Layer3D
SceneViewers are the display windows for a scene.
Scene-Viewer
3DProperties is a layer extension that controls how a layer gets viewed in 3D. Different 3DProperties classes are implemented for different layer types.
ICamera
Camera
I3DProperties
IDataLayerExtension
IPersist
IPersistStream
3DProperties
The camera controls the perspective of a 3D view.
IFeature3DProperties
ICursor
IFeatureCursor
Feature3D-Properties
Raster3D-Properties
Tin3D-Properties
Basic3D-Properties
IMessageDispatcher
Message-Dispatcher
The MessageDispatcher controls navigation ability and cancel tracking while in custom code loops.


TIN and Raster Objects
IDataset
IFunctionalSurface
IGeoDataset
IMetadata
IMetadataEdit
INativeTypeInfo
ISupportErrorInfo
ISurface
ITin
ITinAdvanced
ITinEdit
ITinNodeCollection
ITinSelection
ITinSurface
Tin
The triangulated network object is used for surface modeling and other forms of spatial analysis.
Nodes are the most basic element of a TIN.
ITinNode
TinNode
ITinTriangle
TinTriangle
ITinEdge
TinEdge
Triangles are formed by connected nodes and represent areas.
Edges connect nodes. They can represent linear features.
IClone
IGeoAnalysisEnvironment
IRasterAnalysisEnvironment
IPersist
IPersistStream
Raster-Analysis
Controls analysis environment properties such as extent, cellsize, mask, and output workspace.
Provides support for IDW, kriging, spline interpolation.
IInterpolationOp
IRasterAnalysis-Environment
ISupportErrorInfo
Raster-InterpolationOp
IRasterAnalysis-Environment
IRasterMakerOp
ISupportErrorInfo
Raster-MakerOp
Creates numerically constant or random rasters.
IMathSupportOp
IRasterAnalysis-Environment
RasterMath-SupportOp
Provides basic arithmetic operators such as addition, subtraction, and floating point truncation.
IRasterAnalysis-Environment
IReclassOp
ISupportErrorInfo
Raster-ReclassOp
Reclassifies rasters using tables to define categories.
The 3D Analyst extension object. It is used for licensing and to obtain global raster settings.
IDddEnvironment
IExtension
IExtensionConfig
Ddd-Environment
IRasterAnalysis-Environment
ISupportErrorInfo
ISurfaceOp
Raster-SurfaceOp
Performs surface analysis such as contouring or slope derivation.

When customizing ArcScene you need to have some understanding of its object hierarchy. This helps you determine how to gain access to a particular object that implements the desired functionality.

At the top level of the object hierarchy is the application. From here you can perform tasks related to the application itself, such as opening a new document, or gain access to other objects contained within the application.

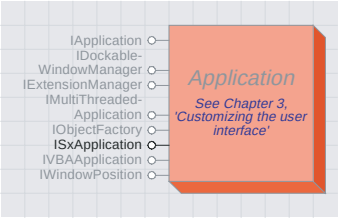
In VBA you can refer to the application directly:

```
Dim pApp as IApplication
set pApp = Application
```

If you're implementing commands and tools in Visual Basic DLLs, the classes get a hook to the application object when they're instantiated:

```
' Declare the application variable at the module level
Private m_pApp as IApplication

Private Sub OnCreate(hook as object)
  Dim m_pApp as IApplication
  set m_pApp = hook
End Sub
```

Once you have a handle on the application, you have access to everything contained within it. Looking at the overview diagram you can see the application is composed of an SxDocument. The document is composed of a scene. The scene has a SceneGraph that references one or more viewers, each of which has a camera. So, if you want access to a viewer's camera you drill down through these objects to gain access to it.



The Application object represents the running ArcScene application.

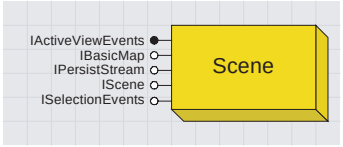
The Application CoClass implements eight interfaces, IApplication, IDockableWindowManager, IExtensionManager, IMultiThreadedApplication, IObjectFactory, ISxApplication, IVbaApplication, and IWindowPosition. Note that ArcMap, ArcCatalog, and ArcScene all implement IApplication. This provides a common foundation from which to build. For example, code using this interface to set the application caption can be identical for all three applications.

```
' Here a module level reference to the application
' is made. Be sure to release it when the class
' terminates (in VB - Class_Terminate).
Private m_pApp as IApplication

Private Sub OnCreate(hook as Object)
  Set m_pApp = hook
End Sub
```

If you'd like to do something that is supported only by the ArcScene application, such as create a new secondary viewer, you must query the application for the ISxApplication interface:

ISxApplication : IUnknown	***Provides access to members that control the ArcScene Application object.***
DefaultBackgroundColor: IRgbColor	*The Default Background Color for New Documents.*
Page: IPage	*The current page settings.*
Paper: IPaper	*The current paper settings.*
Printer: IPrinter	*The current printer settings.*
SelectionEnvironment: ISelectionEnvironment	*The selection environment.*
TOCVisible: Boolean	*Indicates if the TOC is visible.*
ClearGesture	*Clears gesture.*
CopyToClipboard	*Copies the current scene to the Clipboard.*
CreateSubView	*Makes a new viewer.*
Export2d	*Exports the Current Document to a 2D file format.*
Export3d	*Exports the Current Document to a 3D file format.*
RestoreGesture	*Restores gesture.*
ShowContextMenu (in x: Long, in y: Long)	*Displays a context menu for the current selection.*
ShowTOC (in bShow: Boolean)	*Displays the table content.*
SuspendGesture	*Suspends gesture.*

```
' Get the application.
Dim pApp As IApplication
Set pApp = Application

' See if application is ArcScene.
If (TypeOf pApp Is ISxApplication) Then
  Dim pSxApp As ISxApplication
  Set pSxApp = pApp

  ' Create secondary viewer
  pSxApp.CreateSubView
End If
```



The Scene is a container for vector, raster, and graphic data.

This section discusses properties found on the Scene Properties dialog, specifically, vertical exaggeration, background color, spatial reference, area of interest, and illumination. Most of these properties are accessed through the IScene interface that is implemented by the Scene component.

IScene : IUnknown	***Provides access to members that control the scene.***
ActiveGraphicsLayer: ILayer	*The active graphics layer. If no graphic layers exist, a basic memory graphics layer will be created.*
AreaOfInterest: IEnvelope	*The area of interest for the scene.*
BasicGraphicsLayer: IGraphicsLayer	*The basic graphics layer.*
Description: String	*The description of the scene.*
ExaggerationFactor: Double	*The vertical exaggeration of the scene.*
Extent: IEnvelope	*The extent of the scene.*
FeatureSelection: ISelection	*The scene's feature selection.*
Layer (in index: Long) : ILayer	*The layer corresponding to a given index.*
LayerCount: Long	*The number of layers in the scene.*
Layers (uid: IUID, recursive: Boolean) : IEnumLayer	*The layers in the scene of the type specified in the UID. If 'recursive' is true, includes layers in group layers.*
Name: String	*The name of the scene.*
SceneGraph: ISceneGraph	*The scene's scene graph.*
SelectionCount: Long	*The number of selected features.*
SpatialReference: ISpatialReference	*The spatial reference of the scene.*
AddLayer (in pLayer: ILayer, autoArrange: Boolean)	*Adds a layer to the scene.*
AddLayers (in Layers: IEnumLayer, in autoArrange: Boolean)	*Adds multiple layers to the scene, optionally arranging them automatically.*
Applies (in object: IUnknown Pointer) : Boolean	*Indicates if the given object is supported by the scene.*
ClearLayers	*Removes all layers from the scene.*
ClearSelection	*Clears the scene's selection.*
DelayEvents (in delay: Boolean)	*Used to batch operations together in order to minimize system notifications.*
DeleteLayer (in Layer: ILayer)	*Deletes a layer from the scene.*
GetDefaultBackgroundColor (out red: Single, out green: Single, out blue: Single)	*Returns the default background color.*
MoveLayer (in Layer: ILayer, in toIndex: Long)	*Moves a layer to another position within the Table Of Contents.*
ProposeSpatialReference (in pProposedSR: ISpatialReference, out pbChanged: Boolean)	*Proposes a spatial reference for the scene.*
RecalculateExtent	*Forces the scene's extent to be recalculated.*
SelectByShape (in shape: IGeometry, in env: ISelectionEnvironment, in justOne: Boolean)	*Selects features in the scene given a shape and an optional selection environment.*
SelectFeature (in Layer: ILayer, in pFeature: IFeature)	*Selects a feature.*
SetDefaultBackgroundColor (in red: Single, in green: Single, in blue: Single)	*Sets the default background color.*
SuggestExaggerationFactor (in aspectRatio: Double, out ExaggerationFactor: Double)	*Returns the vertical exaggeration factor that achieves the aspect ratio for the scene's extent.*

The one exception, Illumination, is accessed through the ISceneGraph interface that is implemented by the SceneGraph class.

Looking at the object hierarchy you can see that the scene document provides access to the scene. To acquire an IScene interface use the ISxDocument.Document member.

The following example shows how to use the IScene interface to change the scene's vertical exaggeration:

```
' Get the application.
Dim pApp As IApplication
Set pApp = Application


' Get the document from the application.
Dim pSxDoc as ISxDocument
```

```
Set pSxDoc = pApp.Document

' Get the scene from the document.
Dim pScene as IScene
Set pScene = pSxDoc.Scene

' Change the exaggeration.
pScene.ExaggerationFactor = _
  pScene.ExaggerationFactor * 1.5

' Update the display.
pScene.SceneGraph.RefreshViewers
```

Illumination is used to shade areal features and provide visual cues that generate a sense of depth. The light is considered to be at an infinite distance in a particular direction. This position is therefore not defined by an absolute 3D coordinate but through a vector. Changing the direction of the light source requires modifying the SunVector property in ISceneGraph.

The example shows how to use the ISceneGraph interface to change the direction of the light source for illumination:

```
' Get the application.
Dim pApp As IApplication
Set pApp = Application

' Get the document from the application.
Dim pSxDoc as ISxDocument
Set pSxDoc = pApp.Document

' Get the scene from the document.
Dim pScene as IScene
Set pScene = pSxDoc.Scene

' Get the scenegraph from the scene.
Dim pSceneGraph as ISceneGraph
Set pScenegraph = pScene.SceneGraph

' Get the sun vector and change its azimuth (in radians).
Dim pVector as IVector3D
Set pVector = pSceneGraph.SunVector
pVector.Azimuth = pVector.Azimuth + 1.5

' Update the display.
pSceneGraph.RefreshViewers
```

Layers used in ArcScene are the same as those used in ArcMap. Those in ArcScene have additional 3D properties added as a layer extension. If you'd like to add layers to a scene using default properties, the procedure is straightforward. You create a layer, give it a reference to the data, and make a call on the scene to have it added. Default 3D properties will be added automatically.

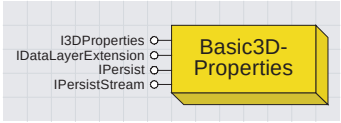
The example shows how to add a layer to a scene:

```
' Construct the layer.
Dim pLayer As IFeatureLayer
Set pLayer = New FeatureLayer
Set pLayer.FeatureClass = pFeatureClass
pLayer.Name = "mylayer"


' Get a reference to the scene.
Dim pSxDoc as ISxDocument
Set pSxDoc = Application.Document
Dim pScene As IScene
Set pScene = pSxDoc.Scene

' Add the layer.
pScene.AddLayer pLayer
```

If you'd prefer to use a methodology that's common to both ArcScene and ArcMap, then use IBasicMap. Both maps and scenes support this interface, so it's only at the highest level where you would have a slight difference in the code acquiring the interface. The rest would be the same.

This example shows how to use the IBasicMap interface, which is common to both ArcScene and ArcMap, to add a layer:

```
Dim pBasicMap As IBasicMap

If (TypeOf pApp Is IMxApplication) Then
  Dim pMxDoc As IMxDocument
  Set pMxDoc = pApp.Document
  Set pBasicMap = pMxDoc.FocusMap
ElseIf (TypeOf pApp Is ISxApplication) Then
  Dim pSxDoc As ISxDocument
  Set pSxDoc = pApp.Document
  Set pBasicMap = pSxDoc.Scene
End If

' Do work and add layer, no need to worry about whether it's a map
or a scene.
.
.
.
pBasicMap.AddLayer pLayer
```



3D Properties is a layer extension that controls how a layer gets viewed in 3D. Different 3DProperties classes are implemented for different layer types.

Layer 3D properties

To control any of the properties for a layer that are found on the Base Heights, Extrusion, or Rendering tabs, you should use the I3DProperties interface, which is implemented by one of several 3D properties classes. The 3D properties classes are layer extensions.

Different layer types support different 3DProperties. For example, raster layers support Raster3DProperties, and feature layers support Feature3DProperties. Regardless of filter type, all support the I3DProperties interface.

I3DProperties : IUnknown	***Provides access to members that control three-dimensional properties.***
BaseExpressionString: String	*The base expression string.*
BaseName: IName	*The name of the base surface.*
BaseOption: tagesriBaseOption	*The base option.*
BaseSurface: IFunctionalSurface	*The base surface.*
DepthPriorityValue: Integer	*The drawing priority to be applied to a layer when in the same location as others.*
ExtrusionExpressionString: String	*The extrusion expression string.*
ExtrusionType: tagesriExtrusionType	*The extrusion type.*
FaceCulling: tagesri3DFaceCulling	*The face culling mode.*
Illuminate: Boolean	*Indicates if areal features are illuminated.*
MaxRasterColumns: Long	*The maximum number of columns for a raster elevation grid.*
MaxRasterRows: Long	*The maximum number of rows for a raster elevation grid.*
MaxTextureMemory: Long	*The maximum texture memory a layer can use.*
OffsetExpressionString: String	*The offset expression string.*
RenderMode: tagesriRenderMode	*The rendering mode.*
RenderRefreshRate: Double	*The rendering refresh rate.*
RenderVisibility: tagesriRenderVisibility	*The render visibility option.*
SmoothShading: Boolean	*Indicates if smooth shading is enabled.*
ZFactor: Double	*The z-factor.*
Apply3DProperties (in owner: IUnknown Pointer)	*Applies 3D properties.*

This example shows how to query a layer for its extensions and acquire the I3DProperties interface:

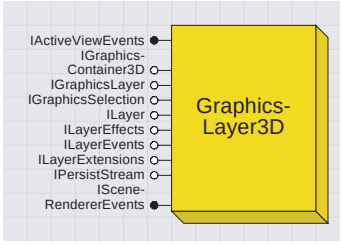
```
' Get the document.
Dim pSxDoc As ISxDocument
Set pSxDoc = Application.Document

' Get the scene from the document.
Dim pScene As IScene
Set pScene = pSxDoc.Scene

' Get a layer in the scene (assumes there is one).
Dim pLayer As ILayer
Set pLayer = pScene.Layer(0)

' Get layer extensions.
Dim pLayerExts As ILayerExtensions
Set pLayerExts = pLayer

' Get 3D properties from extension.
' Layer must have it if it's in a scene.
Dim i As Long
For i = 0 To pLayerExts.ExtensionCount - 1
  If (TypeOf pLayerExts.Extension(i) Is I3DProperties) Then
```

```
    Dim p3DProps As I3DProperties
    Set p3DProps = pLayerExts.Extension(i)
    Exit For
  End If
Next i
```

The following example illustrates how to use I3DProperties to set the base height property for a layer to an expression:

```
' Set the layer's base height property to an expression
' that equals an item multiplied by a constant.
' This case assumes we have a feature layer with an item
' called [SPOT].
p3DProps.BaseOption = esriBaseExpression
p3DProps.BaseExpressionString = "[SPOT] * 3.28"

' Apply the properties and redraw.
p3DProps.Apply3DProperties pLayer
pScene.SceneGraph.RefreshViewers
```

Note that a call to Apply3DProperties may result in the layer being re-loaded from disk into memory. This depends on what properties are changed and if the layer's RenderMode is set to esriRenderCache. This can take some time depending on the amount of data involved. Many 3D properties require a reload. Those that do not are DepthPriorityValue, FaceCulling, Illuminate, RenderRefreshRate, RenderVisibility, and SmoothShading.



Graphics layers are used to render graphic primitives.

Scenes can contain graphics in addition to layers. Graphics are created by tools and commands in ArcScene. They may be created in ArcMap and then pasted into the scene. In order to manage these graphics, ArcScene supports graphics layers.

Every scene has at least one graphics layer, the basic graphics layer. It can be retrieved through IScene's BasicGraphicsLayer property. Querying for the IGraphicsContainer3D interface from the layer gives you the ability, among other things, to add and remove graphic elements. Note that the underlying geometry of added graphic elements should be ZAware (see Geometry section later in this chapter).

IGraphicsContainer3D : IUnknown	***Provides access to members that manipulate the graphics container.***
← Element (in index: Long) : IElement	*The element in the container defined by the given index.*
← ElementCount: Long	*The number of elements in the container.*
← AddElement (in Element: IElement)	*Adds a new graphic element to the container.*
← AddElements (in elements: IElementCollection)	*Adds a collection of new graphic elements to the container.*
← BeginBatchUpdate	*Initiates a batch update of the container.*
← DeleteAllElements	*Deletes all the elements.*
← DeleteElement (in Element: IElement)	*Deletes the given element.*
← EndBatchUpdate	*Terminates a batch update of the container.*
← LocateElements (in pPoint: IPoint, in tolerance: Double) : IEnumElement	*Returns the elements that intersect with the given ray.*
← LocateElementsByEnvelope (in pEnvelope: IEnvelope) : IEnumElement	*Returns the elements that intersect with the given envelope.*
← MoveElementFromGroup (in pGroup: IGroupElement, in pElement: IElement)	*Moves the specified element from the group to the container.*
← MoveElementToGroup (in pElement: IElement, in pGroup: IGroupElement)	*Moves the specified element from the container to the group.*
← Next: IElement	*The next graphic in the container.*
← Reset	*Resets the internal cursor so that 'Next' returns the first element.*

The following example shows how to add a graphic to the basic graphics layer of a scene:

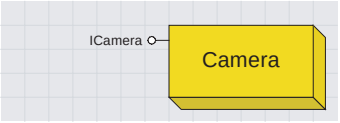
```
' Get the basic graphics layer from the document.
Dim pSxDoc As ISxDocument
Set pSxDoc = Application.Document

Dim pGLayer As IGraphicsLayer
Set pGLayer = pSxDoc.Scene.BasicGraphicsLayer

' Get the container interface from the layer.
Dim pGCon3D As IGraphicsContainer3D
Set pGCon3D = pGLayer

' Add the graphic element. Its geometry should be ZAware.

pGCon3D.AddElement pElement

' Unselect all other elements, select this one and
' show selection handles.
Dim pGS As IGraphicsSelection
Set pGS = pGCon3D
```

```
pGS.UnselectAllElements
pGS.SelectElement pElement

‘ Update the viewers so the change is visible.
pSxDoc.Scene.SceneGraph.RefreshViewers
```

Interactive tools typically require the user to click somewhere in the display. The tool then determines the position of the click and performs an action based on that location. In a 2D map display the user can click anywhere in the display and get x- and y- coordinates whether or not they actually 'hit' something. In a scene, the user must click on something that's displayed, such as a feature or surface, in order to get x, y, and z positional information. This is because a click on the screen defines a ray that starts from the observer position. There are an infinite number of positions along the ray. Only when the ray intersects an object can we return a position.

The following example shows how to use the ISceneGraph interface to locate what a user clicked, if anything, and where (For accurate positional information a triangulated irregular network (TIN) or raster should be clicked on. Other features return their centroid.):

```
' Get the scenegraph.
Dim pSxDoc As ISxDocument
Set pSxDoc = Application.Document
Dim pSG As ISceneGraph
Set pSG = pSxDoc.Scene.SceneGraph

' Get the active viewer. This will be the one the
' user clicked in.
Dim pViewer As ISceneViewer
Set pViewer = pSG.ActiveViewer

' Declare variables that the locate function writes
' results to.
Dim pMapPoint As IPoint
Dim pOwner As stdole.IUnknown
Dim pObject As stdole.IUnknown

' Perform the locate. If you're in a tool's MouseDown event you have
' the x and y screen coordinates where the user clicked. Use the
' locate member to translate those into the 3D point for the
' location of the click and the feature (object) and layer (owner)
' that was clicked on.
pSG.Locate pViewer, x, y, esriScenePickGeography, _
  True, pMapPoint, pOwner, pObject
' Report result.
If (Not pMapPoint Is Nothing) Then
  Dim pLayer As ILayer
  Set pLayer = pOwner
  MsgBox "Layer : " & pLayer.name & vbCrLf & _
         CStr(Round(pMapPoint.x, 3)) & ", " & _
         CStr(Round(pMapPoint.y, 3)) & ", " & _
         CStr(Round(pMapPoint.Z, 3))
Else
  MsgBox "Nothing was clicked on"
End If
```



The camera controls the perspective of a 3D view.

The perspective of a 3D view is defined by a virtual camera. The camera has observer, target, and viewfield properties that can be related to real cameras. The observer represents the position of the camera itself in 3D space. The target is the 3D location the camera is pointing toward. The viewfield angle is like the lens length. When you navigate, changing perspective, you are really manipulating the properties of the camera.

Access to a camera's properties and methods is made through the ICamera interface.

ICamera : IUnknown	***Provides access to members that manipulate the camera.***
Azimuth: Double	*The polar azimuth of the observer relative to the target. When changed, the observer moves.*
Inclination: Double	*The polar inclination of the observer relative to the target. When changed, the observer moves.*
IsUsable: Boolean	*Indicates if the camera has valid parameters.*
MultiPhaseRendering: Boolean	*The state of multiphase rendering. When true, the scene depth can be divided into multiple passes to improve rendering quality.*
Observer: IPoint	*The observer's position.*
OrthoViewingExtent: IEnvelope	*The extent visible by the camera in orthographic view.*
ProjectionType: tagesri3DProjectionType	*The type of projection.*
RollAngle: Double	*The roll angle in degrees.*
Scale: Double	*The orthographic projection scale.*
Target: IPoint	*The target's position.*
UpDirection: IVector3D	*The camera's up-vector.*
VerticalExaggeration: Double	*Adapts the camera to the scene's vertical exaggeration.*
ViewFieldAngle: Double	*The view-field angle in degrees.*
ViewingDistance: Double	*The viewing distance between the observer and the target. When changing, the observer moves.*
CanSeeMBB (in pExtent: IEnvelope) : Boolean	*Indicates if the camera can see any portion of the given extent.*
CanSeeSphere (in pCenter: IPoint, in radius: Double) : Boolean	*Indicates if the camera can see any portion of the given sphere.*
GetIdentifyRay (in dx: Long, in dy: Long) : IRay	*Returns the ray that corresponds to given screen coordinates.*
GetIdentifyVector (in pCursor: IPoint) : IVector3D	*Returns the vector that corresponds to the given screen location.*
HTurnAround (in dAzimuth: Double)	*Turns the camera horizontally around observer by the given azimuth angle.*
LevelObsToTarget	*Levels the observer to the target.*
LevelTargetToObs	*Levels the target to the observer.*
Move (in direction: tagesriCameraMovementType, in factor: Double)	*Moves the camera in the specified direction. The distance is calculated by applying the given factor to the current viewing distance.*
Pan (in startPoint: IPoint, in endPoint: IPoint)	*Moves both the observer and the target so that the object picked as the starting point on-screen assumes the ending-point position.*
PolarUpdate (in distanceFactor: Double, in dAzimuth: Double, in dInclination: Double, in bLimitInclination: Boolean)	*Updates the observer's location by given polar increments.*
PropertiesChanged	*Sets camera's dirty flag.*
QueryDistanceToMBB (in pExtent: IEnvelope, out distance: Double)	*Returns the distance to the given extent.*
QueryDistanceToSphere (in pCenter: IPoint, in radius: Double, out distance: Double)	*Returns the distance to the given sphere.*
QueryDistanceToSphereCenter (in pCenter: IPoint, in radius: Double, out distance: Double)	*Returns the distance to the given sphere's center.*
QueryIdentifyVector (in dx: Long, in dy: Long, pIdentifyVect: IVector3D)	*Returns the vector that corresponds to given screen coordinates.*
QueryOrthoViewingPlatform (in pInExtent: IEnvelope, pOutExtent: IEnvelope, out pScale: Double)	*Returns orthographic projection viewing parameters corresponding to the given extent.*
QueryViewingPlatformMBB (in pExtent: IEnvelope, out pNearPlane: Double, out pFarPlane: Double, out pAngle: Double, out pAspect: Double)	*Returns the viewing parameters corresponding to the given extent.*
QueryViewingPlatformSphere (in pCenter: IPoint, in radius: Double, out pNearPlane: Double, out pFarPlane: Double, out pAngle: Double, out pAspect: Double)	*Returns the viewing parameters corresponding to the given spherical extent.*
RecalcUp	*Updates the up-vector.*
ReplayFrame (in pSceneGraph: ISceneGraph)	*Renders a frame.*
Rotate (in angle: Double)	*Rotates the observer horizontally around the target by a given angle in degrees.*
SetCoordOrigin (in xOrig: Double, in yOrig: Double, in zOrig: Double)	*Adapts the camera to the internal coordinate system of scene graph.*
SetDefaultsMBB (in pExtent: IEnvelope)	*Positions camera so that the entire given extent can be seen.*
SetDefaultsSphere (in pCenter: IPoint, in radius: Double)	*Positions camera so that the entire given spherical extent can be seen.*
Zoom (in ratio: Double)	*Zooms in or out by moving the observer according to the required ratio between the new and previous viewing distances.*
ZoomToRect (in pExtent: IEnvelope)	*Zooms to the given screen extent.*

The following example shows how to use the ICamera interface to change the active viewer's observer position:

```
' Get the scene graph.
Dim pSxDoc As ISxDocument
Set pSxDoc = Application.Document
Dim pScenegraph as ISceneGraph
Set pSceneGraph = pSxDoc.Scene.Scenegraph

' Get the active viewer.
Dim pViewer as ISceneViewer
Set pViewer = pScenegraph.ActiveViewer

' Get the camera.
Dim pCamera as ICamera
Set pCamera = pViewer.Camera

' Get the camera observer.
Dim pObserver as IPoint
Set pObserver = pCamera.Observer

' Move observer to an explicit location.
Dim pExtent As IEnvelope
Set pExtent = pSxDoc.Scene.Extent
pObserver.x = pExtent.XMax
pObserver.y = pExtent.YMax
pObserver.Z = pExtent.zmax

' Set the observer back to the camera
pCamera.Observer = pObserver

' Refresh the viewer display.
pViewer.Redraw true
```

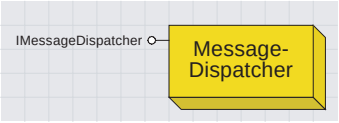
Animation provides a sense of dynamism to the scene. One can animate perspective, the visibility of layers, or the graphics of a scene. More than one type of animation can be occurring at the same time. Animation lets a scene drive itself, requiring no human interaction.

Animating perspective (fly-through)

A classic form of animation is the fly-through where one has the sense they are being flown through the scene. The basic concept involves moving a viewer's camera along a predefined flight path at a given step interval and refreshing the display at each step.

The following example shows how to use the ICamera interface to move the camera in a loop to simulate a fly-through:

```
' Get pre-defined flight path (selected 1 part
' polyline).
..
..

' Get the viewer.
Dim pViewer As ISceneViewer
Set pViewer = pScene.SceneGraph.ActiveViewer

' Get the camera from the viewer.
Dim pCamera As ICamera
Set pCamera = pViewer.Camera

' Get pointcollection interface from polyline
' representing flight path.
Dim pPC As IPointCollection
Set pPC = pPolyline

' Set count variable equal to number vertices on
' flight path.
Dim count As Long
count = pPC.PointCount

' Instantiate observer and target objects as
' points.
Dim pObs As IPoint
Set pObs = New Point
Dim pTar as Ipoint
Set ptar = New Point

' In a loop, move camera from one vertex to the
' next have it looking ahead a couple vertices
' (looking ahead provides a more natural feel).
Dim i As Long
For i = 0 To count - 3
  pPC.QueryPoint i, pObs
  pCamera.Observer = pObs
```



The MessageDispatcher controls navigation ability and cancel tracking while in custom code loops.

```
  pPC.QueryPoint i + 2, pTar
  pCamera.Target = pTar
  pViewer.Redraw True
Next I
```

Something that may come into play here is the need for interrupt support. The user should be able to escape at any time, especially if the animation contains a long flight path or takes a long time to run through for any reason. This support can be incorporated through the use of a MessageDispatcher object.

The MessageDispatcher implements an IMessageDispatcher interface that has properties and methods to support the handling of Windows messages (events) within a loop.

IMessageDispatcher : IUnknown	***Provides access to members for manipulating message queue and keeping track of cancellation.***
Cancelled (in hWnd: Long) : Boolean	*Indicates if a cancel action has been conducted.*
CancelOnClick: Boolean	*Indicates if a mouse click is seen as a cancel action.*
CancelOnEscPress: Boolean	*Indicates if pressing the escape key is seen as a cancel action.*
Dispatch (in hWnd: Long, in bSingle: Boolean, pbCancelled: Variant)	*Dispatches messages associated with the window.*
Remove (in hWnd: Long, in bSingle: Boolean, pbCancelled: Variant)	*Removes messages associated with the window.*

In the loop at the end of the previous code example, we moved a camera along the vertices of a polyline. A change to that loop that keeps the camera flying along the path and listens for a user escape (via the ESC key) would be implemented as follows:

```
' Loop using MessageDispatcher.
Dim pMD As IMessageDispatcher
Set pMD = New MessageDispatcher
pMD.CancelOnClick = False
pMD.CancelOnEscPress = True
Dim bDone As Boolean
bDone = False
Do While (Not bDone)
  pPC.QueryPoint i, pObs
  pCamera.Observer = pObs
  pPC.QueryPoint i + 2, pTar
  pCamera.Target = pTar
  pViewer.Redraw True
  i = i + 1
  If (i = (count - 3)) Then
    bDone = True
  Else
    ' Cancelled member allows no interaction by user
    ' while in this loop except to escape.
    bDone = pMD.Cancelled(pViewer.hWnd)
  End If
Loop
```

The loop will continue until either the end of the polyline is reached or an ESC is found in the message queue. In this case, the

MessageDispatcher only listens for an escape to cancel and does not dispatch any other events. This does not permit navigation or use of other tools while in the loop.

The loop will continue until either the end of the polyline is reached or a ESC is found in the message queue. The MessageDispatcher is listening for the escape key. In this case, the Remove function is used to eliminate other messages from the event queue so any attempt from the user to navigate is ignored.

Here's another example of an animation loop. This time it simply rotates the camera's observer about the target, making it appear as though the scene is spinning. The MessageDispatcher is also set to dispatch events so the user can navigate even though we're inside the loop:

```
' Get the Camera from the active viewer.
Dim pSxDoc As ISxDocument
Set pSxDoc = ThisDocument
Dim pViewer As ISceneViewer
Set pViewer = pSxDoc.Scene.SceneGraph.ActiveViewer
Dim pCamera As ICamera
Set pCamera = pViewer.Camera

' Create a dispatcher to handle navigation/cancel events
Dim pDispatch As IMessageDispatcher
Set pDispatch = New MessageDispatcher
pDispatch.CancelOnClick = False
pDispatch.CancelOnEscPress = True
Dim bCancel As Boolean
bCancel = False

' Loop until user presses ESC key
Do While (Not bCancel)
  pCamera.Rotate 2
  pViewer.Redraw True
  pDispatch.Dispatch pViewer.hWnd, False, bCancel
Loop
```

To make the animation fast, smooth, and jitter free, one must pay attention to making a scene that can refresh itself quickly and to having a smooth path to move the camera along. If there's a lot of data to display and the animation is too slow to be viewed in real time, then consider making a video. To do this take an image snapshot at each step in the loop that repositions the camera. Consider the snapshots a sequence of frames in a movie. Other software can be used to take the collection of frames and compile them into a digital video in MPEG, AVI, or QuickTime format.

Animating through a set of layers

If one has data collected for a location over time, where each step in time can be represented by a different layer (e.g., weekly average ocean tem-

perature over the course of a year), these can be 'played' in sequence to animate change through time. The idea is to have only one layer turned on at a time, starting with the first and incrementing to the next until the end is reached.

The following example shows how to display a set of layers in sequence:

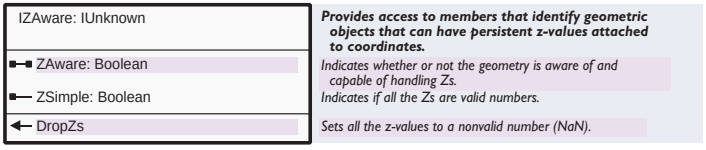
```
' Get the scene graph.
Dim pSxDoc As ISxDocument
Set pSxDoc = ThisDocument
Dim pScene As IScene
Set pScene = pSxDoc.Scene
Dim pSG As ISceneGraph
Set pSG = pScene.SceneGraph

' Turn off all layers. Note, this isn't done through
' the table of contents (TOC) but rather the scene
' graph as it has a faster method to control a layer's
' visibility.
Dim i As Long
For i = 0 To (pScene.LayerCount - 1)
  Dim pLayer As ILayer
  Set pLayer = pScene.Layer(i)
  pSG.SetOwnerVisibility pLayer, False
Next i

' Create a MessageDispatcher to handle events like
' navigation and Esc.
Dim pMD As IMessageDispatcher
Set pMD = New MessageDispatcher
pMD.CancelOnClick = False
pMD.CancelOnEscPress = True

Dim bCanceled As Boolean
bCanceled = False
Dim curLayer As Long
curLayer = 0

' Loop progressively through list of layers
' turning on the visibility of one while
' turning off the visibility of the previous.
' Loop until user presses ESC.
Do While (Not bCanceled)
  ' Turn on current layer.
  Set pLayer = pScene.Layer(curLayer)
  pSG.SetOwnerVisibility pLayer, True

  ' Turn off previous layer.
  If (curLayer = 0) Then
    Set pLayer = pScene.Layer(pScene.LayerCount - 1)
    pSG.SetOwnerVisibility pLayer, False
```

```
    Else
      Set pLayer = pScene.Layer(curLayer - 1)
      pSG.SetOwnerVisibility pLayer, False
    End If

    ' Redraw.
    pSG.RefreshViewers

    ' Permit user to navigate.
    pMD.Dispatch pSG.ActiveViewer.hWnd, False, bCanceled

    ' Increment.
    curLayer = curLayer + 1
    If (curLayer = pScene.LayerCount) Then
      curLayer = 0
    End If
  Loop

    End If
  Loop
```

The core ArcGIS framework supports the creation and storage of 3D geometry. A 3D Analyst license is not required for this. 3D Analyst is a primary client of this core 3D support though, so it's worth having some discussion on the topic here.

ZAwareness and storage of 3D features

ZAwareness is a property on the geometry of a feature. It has several purposes. First, it's used to indicate whether z-values should be stored when the feature is persisted. If the GeomDef of the FeatureClass being used to store the feature has its HasZs property set to true, then all feature geometries in the FeatureClass must be ZAware. Secondly, any geometry operation other than simple reading and writing of z-values requires ZAwareness to be true for z's to be maintained. Additionally, geometries used for graphics in ArcScene or the SceneViewer Control need to be ZAware.



The following example shows how to create a 3D point and make it ZAware:

```
' Create a 3D point.
Dim pPoint as IPoint
Set pPoint = New Point

' Assign coordinates.
pPoint.X = 10
pPoint.Y = -5
pPoint.Z = 2.33

' Make it ZAware.
```

```
Dim pZAware as IZAware
Set pZAware = pPoint
pZAware.ZAware = True
```

Manipulating 3D geometries

Geometry provides some tools for working with 3D geometries. The IZCollection and IZ interfaces can be used on collections. They provide a variety of basic functions such as determining the Z range of a geometry or multiplying all its vertex heights by a constant.

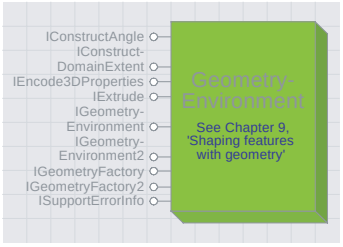
IZCollection: IUnknown	*Provides access to members that identify geometric collection objects that can have z-values attached to coordinates and defines operations on such objects.*
ZMax: Double	*The maximum z-value.*
ZMin: Double	*The minimum z-value.*
MultiplyZs (factor: Double)	*Multiplies all the z-values by a factor.*
OffsetZs (Offset: Double)	*Offsets all the z-values by an offset value.*

IZ: IZCollection	*Provides access to members that identify geometric objects that can have 3D coordinates and defines operations on such objects.*
ZVertical: Boolean	*Indicates if at least two consecutive vertices of this polyline or polygon have the same x and y values but distinct z-values.*
CalculateNonSimpleZs	*Calculates the non-simple z-values by extrapolation/ interpolation.*
InterpolateFromSurface (interpolationSurface: IFunctionalSurface)	*Uses the specified functional surface to generate z-values for the vertices of this object.*
InterpolateZsBetween (startPart: Long, StartPoint: Long, endPart: Long, EndPoint: Long)	*Generates z-values by linear interpolation for all vertices in the range [start+1, end-1].*
SetConstantZ (zLevel: Double)	*Sets z-coordinates at all locations on this object equal to a single value.*

This example will offset a polyline's z-values by a constant amount:

```
' This assumes the polyline vertices have been
' assigned Z values.
Dim pZ As IZ
Set pZ = pPolyline
pZ.OffsetZs 100
```

Smooth shading and texture mapping of multipatches



The geometry environment controls the environment in which geometric objects are created.

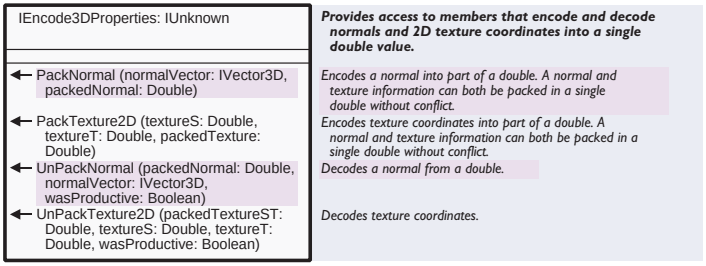
A Multipatch is a type of geometry used to provide a boundary representation for 3D features. Multipatches can be used to represent simple things such as cylinders, cones, and spheres, as well as more complex objects such as buildings. Refer to the online help for Multipatches to understand their basic properties and how to construct them. By default, multipatches are rendered as a collection of flat shaded faces, but it's possible to place images on them and to draw them using smooth shading.

To place an image on a multipatch, one must first provide instructions on how to place the image on the geometry. This is accomplished by assigning texture coordinates to each vertex of the mutipatch. These coordinates, referred to as *s* and *t*, have values that range from 0 to 1, represent-

ing the beginning to end of the image in x and y directions.

Once *s* and *t* are known for a vertex, the GeometryEnvironment is used to encode them into the vertex's m (measure)-value.

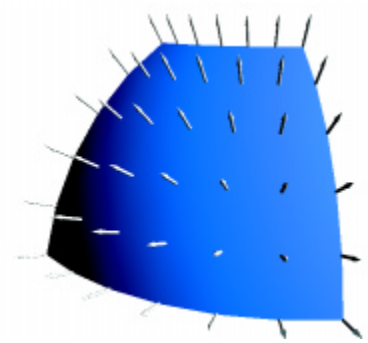
The GeometryEnvironment is used for properties that are not specific to, or are implemented by, an individual geometry. One thing it does that's useful for 3D is implement an interface to encode and decode *s* and *t* coordinate values for texture mapping and normals for smooth shading.



This example shows how to apply texture coordinates to a multipatch vertex:

```
' Acquire interface from the geometry environment to
' encode texture coordinates.
Dim pPack As IEncode3DProperties
Set pPack = New GeometryEnvironment

' Pack s/t into a double and assign as measure to
' point that will be used as vertex for multipatch.
Dim m As Double
m = 0
pPack.PackTexture2D s, t, m
pPoint.m = m
```

Smooth shading of multipatches is useful when one wants to hide abrupt changes present in coarse geometry and make it appear as though the geometry is more complex and continuous. An example where one might like to use smooth shading is a sphere where you'd like to use as little geometry as possible to improve rendering performance, thus causing discontinuities, but have it appear perfectly round. Smooth shading is accomplished by assigning vector normals to each vertex of a multipatch.



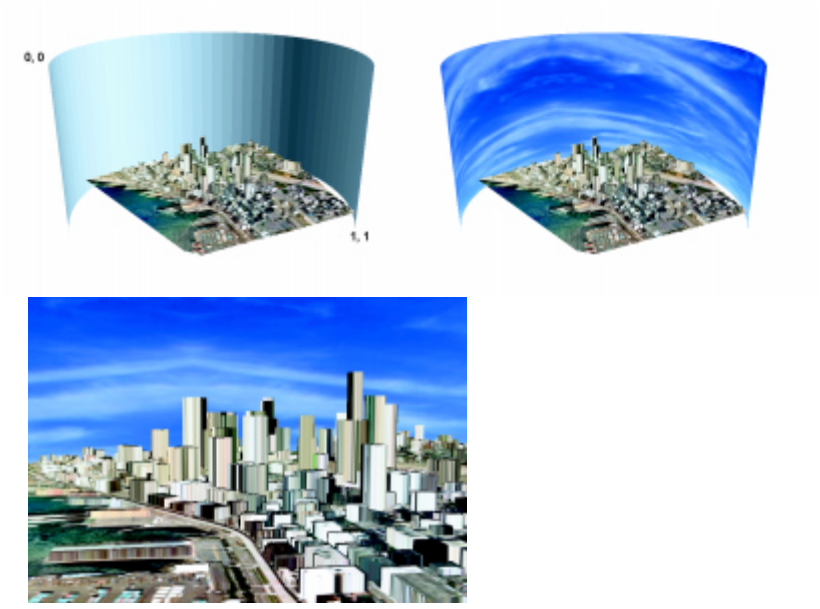
The next example illustrates the use of IEncode3DProperties to encode normals onto multipatch vertices for smooth shading:

```
' Acquire interface to encode vector normal
' from geometry environment.
Dim pPack As IEncode3DProperties
Set pPack = New GeometryEnvironment

' Create a vector at the location of a multipatch
' vertex. Its direction is typically orthogonal
' (pointing away from) the object at that location.
Dim pVector As IVector3D
Set pVector = New Vector3D

' Assign directional values.
pVector.XComponent = -5
pVector.YComponent = 2
pVector.ZComponent = 2.3

' Make sure its normalized (its magnitude is 1.0).
pVector.Normalize

' Pack vector into a double and assign as measure
' to point that will be used as vertex for
' multipatch.
Dim m as Double
m = 0
pPack.PackNormal pVector, m
pPoint.m = m
```

Note that both texture coordinates and normals can be packed into the same m-value. This supports smoothly shaded geometry that is also texture mapped.

```
m = 0
pPack.PackTexture2D s, t, m
pPack.PackNormal pVector, m
pPoint.m = m
```

Images can be used as symbols for features represented by MultiPatches. The technique of mapping imagery to geometry is known as texture mapping. The image is a texture that gets mapped to the geometry using the texture coordinates stored for each vertex of the geometry.



This example applies a PictureFill symbol as a texture to a multipatch-based feature layer:

```
' Get the scene.
Dim pSxDoc As ISxDocument
Set pSxDoc = Application.Document

Dim pScene As IScene
Set pScene = pSxDoc.Scene

' Get the renderer for a layer. This code assumes
' the first layer in the doc contains a multipatch
' with texture coordinates and uses a simple renderer
' that has one symbol. An example might be a multipatch
' based sphere used to represent the Earth.

Dim pFLayer As IFeatureLayer
Set pFLayer = pScene.Layer(0)

Dim pGFLayer As IGeoFeatureLayer
Set pGFLayer = pFLayer
```

```
Dim pSRenderer As ISimpleRenderer
Set pSRenderer = pGFLayer.Renderer

‘ Create a PictureFillSymbol to use as texture.
Dim pPFSymbol As IPictureFillSymbol
Set pPFSymbol = New PictureFillSymbol
pPFSsymbol.CreateFillSymbolFromFile _
  esriIPictureBitmap, _
  “\\leo\LakeTahoe\squaw_valley\clouds.bmp”

‘ Assign symbol to renderer.
Set pSRenderer.Symbol = pPFSymbol

‘ Update the display.
Dim pSceneGraph As ISceneGraph
Set pSceneGraph = pScene.SceneGraph
pSceneGraph.Invalidate pFLayer, True, False
pSceneGraph.RefreshViewers
```

3D Analyst ships with an ActiveX viewer control. The control is used to provide 3D viewing capabilities in ActiveX control containers such as Microsoft® Word and Microsoft® PowerPoint, custom applications, and custom commands added to ArcMap and ArcScene. It is capable of displaying Scene documents, layers, and graphics. It can be customized in much the same way as an ArcScene viewer.

3D Analyst must be installed on the same machine the SceneViewer control is to be used on.

Adding the SceneViewer control to a Microsoft® PowerPoint presentation

1. Start PowerPoint and open the Control Toolbox toolbar.
2. Click on the 'More Controls' button of the Toolbox toolbar.
3. From the dropdown menu select ESRI SceneViewer Control.
4. Draw a rectangle on the slide where you'd like the SceneViewer control placed.
5. Right-click on the control and select Properties.
6. Select the DocName property and press its browse button to reference an existing Scene document.
7. Switch PowerPoint into presentation mode and view the slide.
8. After the scene is loaded it will display, and you can use the mouse to navigate.

Note that when you switch back to design mode, the scene is unloaded from memory and needs some time to reload when the slide is viewed again in presentation mode. Before presenting the slide show to an audience, you should switch PowerPoint to presentation mode, view the slide containing the SceneViewer (letting it load the scene in memory), then go back to the first slide to start. This caches the scene and ensures the slide containing the viewer will appear immediately when the slide is displayed.

Adding the SceneViewer control to a Microsoft® Visual Basic form

1. Start Visual Basic and open the Components dialog.
2. From the Controls tab select ESRI SceneViewer Control. This will add a SceneViewer icon to the Toolbox toolbar.
3. Select the SceneViewer icon on the Toolbox.
4. Draw a rectangle on the form where you'd like the SceneViewer control placed.
5. Switch to the form's Code View and select the SceneViewerCtrl in the Object dropdown combo.
6. Select events that need to be handled from the Procedure dropdown combo.

To populate the control with data, one can either set the DocName property to reference a Scene document or get a handle on the control's scene and use it to add layers or graphics.

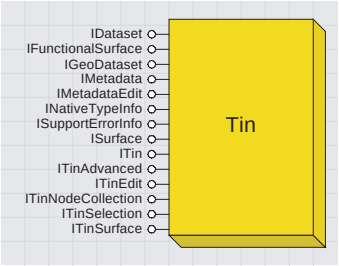
```
' Reference a scene document
SceneViewerCtrl1.DocName = "c:\temp\myscene.sxd"
```

or

```
' Add a layer
SceneViewerCtrl.SceneGraph.Scene.AddLayer pLayer
```

or

```
' Add a graphic element
Dim pMap As IBasicMap
Set pMap = SceneViewerCtrl1.SceneGraph.Scene
Dim pGCon As IGraphicsContainer3D
Set pGCon = pMap.BasicGraphicsLayer
Set pGCon = pGLayer
pGCon.AddElement pElement
Viewer.Redraw False
```



The triangulated irregular network object is used for surface modeling and other forms of spatial analysis.

3D Analyst supports two types of surface models: TINs and rasters. In general, each has its own methodology for creation, data structure access, and analysis. They both support the ISurface interface though, so a number of surface analysis functions can be performed in an identical fashion on both.

Accessing an existing TIN

There are several approaches for accessing an existing TIN dataset. These include going through the TinWorkspaceFactory, using the ITinAdvanced interface, and getting the TIN from a TinLayer.

Here's an example to access a TIN using the workspace factory:

```
' Declare and set variables for the TIN path and name.
Dim sDir As String
sDir = "c:\mytinworkspace"

Dim sName As String
sName = "mytin"

' Create a TIN workspace factory.
Dim pWSFact As IWorkspaceFactory
Set pWSFact = New TinWorkspaceFactory

' Test for and open the TIN if valid
If (pWSFact.IsWorkspace(sDir)) Then
  Dim pTinWS As ITinWorkspace
  Set pTinWS = pWSFact.OpenFromFile(sDir, 0)
  If (pTinWS.IsTin(sName)) Then
    Dim pTin As ITin
    Set pTin = pTinWS.OpenTin(sName)
  End If
End If
```

Here's an example using the ITinAdvanced interface:

```
' CoCreate a TIN object
Dim pTinAdv As ITinAdvanced
Set pTinAdv = New Tin

' Have it reference a TIN on disk
pTinAdv.Init "c:\mytinworkspace\mytin"
```

The following example gets a TIN from a TinLayer:

```
' Get the scene document (it could just as easily be a map).
Dim pSxDoc As ISxDocument
Set pSxDoc = Application.Document

' Get IBasicMap
Dim pBasicMap As IBasicMap
Set pBasicMap = pSxDoc.Scene ' in MX this would come from FocusMap

' Find the first TIN layer in the scene and get a
```

```
' reference to its dataset.
Dim layerInx As Long
For layerInx = 0 To pBasicMap.LayerCount - 1
  Dim pLayer As ILayer
  Set pLayer = pBasicMap.Layer(layerInx)
  If (TypeOf pLayer Is ITinLayer) Then
    Dim pTinLayer As ITinLayer
    Set pTinLayer = pBasicMap.Layer(layerInx)
    Dim pTinAdv As ITinAdvanced
    Set pTinAdv = pTinLayer.Dataset
    Exit For
  End If
Next layerInx
```

Creating a new TIN

TINs are created and edited through the ITinEdit interface. This interface provides functions for adding entire feature classes and individual shapes to the triangulation. There's also the ability to work with individual elements, to set z-values nodes, to change edge types, to mask or unmask triangles, and many more options. The interface is queried directly off the TIN object.

ITinEdit : IUnknown	*Provides access to members that control TIN editing.*
IsDirty: Boolean	*Indicates if the TIN is in edit mode.*
IsEditable: Boolean	*Indicates if the specified feature can be edited.*
IsInEditMode: Boolean	*Indicates if the specified feature is in an editable mode.*
AddFromFeatureClass (in pFeatureClass: IFeatureClass, pFilter: IQueryFilter, in pHeightField: IField, in pValueField: IField, in Type: tagesriTinSurfaceType, pbUseShapeZ: Variant)	*Adds features from a feature class to the TIN.*
AddFromFeatureCursor (in pCursor: IFeatureCursor, in pHeightField: IField, in pValueField: IField, in Type: tagesriTinSurfaceType, pbUseShapeZ: Variant)	*Adds features from a feature cursor to the TIN.*
AddFromPixelBlock (in xOrigin: Double, in yOrigin: Double, in xPixelSize: Double, in yPixelSize: Double, in valueForNoData: Variant, in block: Variant, in zTolerance: Double, pMaxPoints: Variant, pbToleranceAchieved: Variant)	*Adds pixels from a pixel block to the TIN.*
AddPointZ (in pPoint: IPoint, in value: Long) : Long	*Adds a 3D point to the TIN.*
AddShape (in pShape: IGeometry, in Type: tagesriTinSurfaceType, in value: Long, pZ: Variant)	*Adds a 2D shape to the TIN.*
AddShapeZ (in pShape: IGeometry, in Type: tagesriTinSurfaceType, in value: Long, pbUseShapeZ: Variant)	*Adds a 3D shape to the TIN.*
AddWKSPointZ (in pPoint: _WKSPointZ, in vlue: Long) : Long	*Adds a well-known structure point to the TIN.*
DeleteNode (in Index: Long)	*Deletes a specified node from the TIN.*
DeleteNodesOutsideDataArea	*Deletes all nodes from outside the TIN interpolation zone.*
DeleteNodeTagValues	*Deletes all node tag values in the TIN.*
DeleteSelectedNodes	*Deletes specified nodes from the TIN.*
DeleteTriangleTagValues	*Deletes all triangle face tag values in the TIN.*
InitNew (in pExtent: IEnvelope)	*Initializes a new TIN using the passed extent to define the data area.*
PropagateTriangleTagValue (in pSeed: ITinTriangle, in value: Long, in bStopAtEnforcedEdge: Boolean)	*Propagates triangle tag value changes to all immediate triangles with the same initial value.*
Refresh	*Updates all TIN values based on current edits.*
Save	*Saves edits to disk.*
SaveAs (in newName: String, pOverWrite: Variant)	*Saves the TIN to disk using the specified name.*
SetEdgeType (in Index: Long, in Type: tagesriTinEdgeType)	*Sets the type of the triangle edge referenced by the TIN.*
SetNodeTagValue (in Index: Long, in value: Long)	*Sets the tag value of a TIN node referenced by the index.*
SetNodeZ (in Index: Long, in Z: Double)	*Sets the z-value of a TIN node referenced by the index.*
SetSpatialReference (in pSpatialReference: ISpatialReference)	*Sets a copy of the specified spatial reference to the TIN.*
SetTriangleInsideDataArea (in Index: Long)	*Sets a triangle within the TIN interpolation zone.*
SetTriangleOutsideDataArea (in Index: Long)	*Sets a triangle outside of the TIN interpolation zone.*
SetTrianglesInsideDataArea	*Sets all triangles within the TIN interpolation zone.*
SetTriangleTagValue (in Index: Long, in value: Long)	*Sets the face tag value of the triangle referenced by the index.*
StartEditing: Boolean	*Initiates edit mode.*
StopEditing (in bSaveEdits: Boolean) : Boolean	*Terminates edit mode, optionally saving changes to disk.*

When creating a new TIN you need to know what its approximate data extent will be and optionally give it a spatial reference. This information typically comes from the feature data that will be used in the triangulation process.

This example shows how to create a new TIN:

```
' Get the extent and spatial reference of the
' feature class that will be used to build the TIN.
' Note, you could be using more than one feature
' class, if so you'll want to get the union of
' their extents.
Dim pGeoDataset As IGeoDataset
Set pGeoDataset = pFeatureClass

Dim pExtent As IEnvelope
Set pExtent = pGeoDataset.Extent

' The TIN will adopt an extent's spatial reference if it's set.
set pExtent = pGeoDataset.SpatialReference

' CoCreate a TIN object and get an ITinEdit
' interface from it.
Dim pTinEdit As ITinEdit
Set pTinEdit = New Tin

' Initialize the TIN
pTinEdit.InitNew pExtent
```

Adding data to a TIN

Data can be added to a TIN either one shape at a time or from a feature class. When many shapes are involved, a feature class, or a collection of feature classes, is typically used. This is for the sake of performance and convenience. Use of individual shapes is most useful for something like an interactive TIN editor. Data can be added to an existing TIN or one you just created.

The ITinEdit.AddFromFeatureClass member is used to add features from a feature class. This member requires a reference to the feature class and an enumeration that indicates how those features are to be added to the triangulation. Other parameters let you use a subset of the input features, specify the height source, and tag values. These can be set to 'Nothing' if they aren't to be used. If the features to be added have z-values and you'd like those to be used for height, then pass the shape field for the height field argument. If you pass 'Nothing' for the height field, then the features will have their heights interpolated off the existing state of the TIN. This implies other features with heights must already have been added.

Here are three methods used to add data to a TIN:

```
' Call for adding shapes from a feature class
pTinEdit.AddFromFeatureClass pFeatureClass, pFilter, pHeightField, _
  pTagValueField, esriTinMassPoint

' Call for adding a 2D shape. The '0' indicates no tag value.
pTinEdit.AddShape pPolyline, esriTinZLessHardLine, 0

' Call for adding a 3D shape. The '0' indicates no tag value.
pTinEdit.AddShapeZ pPolyline, esriTinHardLine, 0
```

Accessing an existing raster

There are a couple approaches for accessing an existing raster dataset. These include going through the RasterWorkspaceFactory and getting the raster from a RasterLayer.

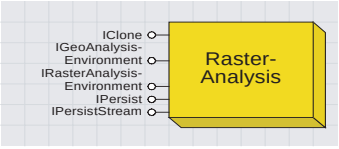
Here's how to access a raster using the workspace factory:

```
' Declare and set variables for the raster path and
' name.
Dim sDir As String
sDir = "c:\myrasterworkspace"

Dim sName As String
sName = "myraster"

' Create a raster workspace factory.
Dim pWSFact As IWorkspaceFactory
Set pWSFact = New RasterWorkspaceFactory

' Open the raster.
If (pWSFact.IsWorkspace(sDir)) Then
  Dim pRasterWS As IRasterWorkspace
  Set pRasterWS = pWSFact.OpenFromFile(sDir, 0)
  Dim pRasterDataset As IRasterDataset
  Set pRasterDataset = pRasterWS.OpenRasterDataset(sName)
End If
```

The following example uses a raster layer:

```
' Get the Scene doc (it could just as easily be a map).
Dim pSxDoc As ISxDocument
Set pSxDoc = Application.Document

' Get IBasicMap
Dim pBasicMap As IBasicMap
Set pBasicMap = pSxDoc.Scene ' in MX this would come from FocusMap

' Find the first raster layer in the scene and get a
' reference to its dataset (for the first band).
Dim layerInx As Long
For layerInx = 0 To pBasicMap.LayerCount - 1
  Dim pLayer As ILayer
```

```
  Set pLayer = pBasicMap.Layer(layerInx)
  If (TypeOf pLayer Is IRasterLayer) Then
    Dim pRasterLayer As IRasterLayer
    Set pRasterLayer = pBasicMap.Layer(layerInx)
    Dim pRaster As IRaster
    Set pRaster = pRasterLayer.Raster
    Dim pRasterBands As IRasterBandCollection
    Set pRasterBands = pRaster
    Dim pRasterBand As IRasterBand
    Set pRasterBand = pRasterBands.Item(0)
    Dim pRasterDataset As IRasterDataset
    Set pRasterDataset = pRasterBand.RasterDataset
    Exit For
  End If
Next layerInx
```



The raster analysis object controls analysis environment properties such as extent, cellsize, mask, and output workspace.

Raster analysis environment

The raster analysis environment is used to define the output workspace, extent, cellsize, and mask for results performed by spatial operators on rasters.

The IRasterAnalysisEnvironment interface provides access to set these properties.

IRasterAnalysisEnvironment : IGeoAnalysisEnvironment	***Provides access to members that control the environment for raster analysis.***
DefaultOutputRasterPrefix: String	*The default output raster prefix.*
DefaultOutputVectorPrefix: String	*The default output vector prefix.*
Mask: IGeoDataset	*Mask allows processing to occur only for a selected set of cells.*
VerifyType: esriRasterVerifyEnum	*The verify type of the RasterAnalysis.*
GetCellSize (out envType: esriRasterEnvSettingEnum, out Cellsize: Double)	*Gets the type and value of cell size in the RasterAnalysis.*
GetExtent (out envType: esriRasterEnvSettingEnum, out Extent: IEnvelope)	*Gets the type and values of extent in the RasterAnalysis.*
Reset	*Removes all previously stored default RasterAnalysis environments.*
RestoreToPreviousDefaultEnvironment	*Restores to the previous default RasterAnalysis environment.*
SetAsNewDefaultEnvironment	*Sets the raster analysis environment of the object as new default environment.*
SetCellSize (in envType: esriRasterEnvSettingEnum, cellSizeProvider: Variant)	*Sets the type and value of cell size in the RasterAnalysis.*
SetExtent (in envType: esriRasterEnvSettingEnum, extentProvider: Variant, snapRasterData: Variant)	*Sets the type and values of extent in the RasterAnalysis.*

The following code is used to obtain the analysis environment as defined by the Raster Options dialog found in ArcMap and ArcScene.

```
‘ Get ArcMap/ArcScene raster analysis environment
‘ from the 3D extension object.
Dim pDDDEnv As IDddEnvironment
Set pDDDEnv = Application.FindExtensionByName(“3D Analyst”)
Dim pRasterEnv As IRasterAnalysisEnvironment
Set pRasterEnv = pDDDEnv.GetRasterSettings
```

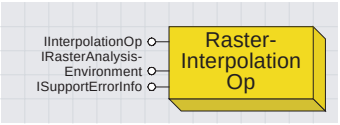
If your code will be running in ArcMap or ArcScene, you have the choice

of using this or instantiating your own analysis object. If your code is in a different application, you must instantiate your own.

Creating a new raster surface

Creation of a raster surface is accomplished through interpolation. The RasterInterpolationOp class provides access to a number of interpolation techniques.

A look at the IInterpolationOp interface reveals support for IDW, spline, trend, and kriging with related variogram methods.



The raster interpolation op object provides support for IDW, kriging, and spline interpolation.

IInterpolationOp : IUnknown	*Provides access to members that control the interpolating of a Geodataset.*
IDW (in geoData: IGeoDataset, in power: Double, in radius: IRasterRadius, barrier: Variant) : IGeoDataset	*Interpolates using IDW.*
Krige (in geoData: IGeoDataset, in semiVariogramType: esriGeoAnalysisSemiVariogramEnum, in radius: IRasterRadius, in outSemiVariance: Boolean, barrier: Variant) : IGeoDataset	*Interpolates using kriging.*
Spline (in geoData: IGeoDataset, in splineType: esriGeoAnalysisSplineEnum, weight: Variant, numPoints: Variant) : IGeoDataset	*Interpolates using splining.*
Trend (in geoData: IGeoDataset, in trendType: esriGeoAnalysisTrendEnum, in order: Long) : IGeoDataset	*Interpolates using trend surface.*
Variogram (in geoData: IGeoDataset, in semiVariogram: IGeoAnalysisSemiVariogram, in radius: IRasterRadius, in outSemiVariance: Boolean, barrier: Variant) : IGeoDataset	*Interpolates using Variogram.*

The following code takes a point layer and creates a raster using IDW:

```
' Get the IGeoDataset interface from a point layer.
Dim pInputGDS As IGeoDataset
Set pInputGDS = pLayer.FeatureClass

' Get the workspace of the input data. The output
' from IDW will be placed here.
Dim pDS As IDataset
Set pDS = pInputGDS
Dim pWorkspace As IWorkspace
Set pWorkspace = pDS.Workspace

' Create a new raster interpolation operator.
Dim pInterpOp As IInterpolationOp
Set pInterpOp = New RasterInterpolationOp

' Get the operator's analysis environment and set
' pertinent properties.
Dim pOpAnalEnv As IRasterAnalysisEnvironment
Set pOpAnalEnv = pInterpOp
Set pOpAnalEnv.OutWorkspace = pWorkspace
pOpAnalEnv.SetCellSize esriRasterEnvValue, _
  (pInputGDS.Extent.Width / 250)
```

```
pOpAnalEnv.SetExtent esriRasterEnvValue, pInputGDS.Extent

' IDW has a radius parameter, create and set
' properties for it.
Dim pRadius As IRasterRadius
Set pRadius = New RasterRadius
pRadius.SetVariable 8, pInputGDS.Extent.Width

' Make a descriptor from the input dataset. This
' code assumes the input has a field called "SPOT"
' to provide heights.
Dim pFilt As IQueryFilter
Set pFilt = New QueryFilter
pFilt.SubFields = "*"
Dim pFCDesc As IFeatureClassDescriptor
Set pFCDesc = New FeatureClassDescriptor
pFCDesc.Create pInputGDS, pFilt, "SPOT"

' Execute the IDW interpolant.
Dim pOutGDS As IGeoDataset
Set pOutGDS = pInterpOp.IDW(pFCDesc, 2, pRadius)

' Add the result as a layer to the scene.
' Raster operators produce 'temporary' output
' so you must continue using it or save it.
' Otherwise it will be deleted automatically.
Dim pRasterLayer As IRasterLayer
Set pRasterLayer = New RasterLayer
pRasterLayer.CreateFromRaster pOutGDS
```

```
pRasterLayer.name = "idw"
pScene.AddLayer pRasterLayer
```

Surface analysis

Surface analysis functions common to both TINs and rasters are implemented in the ISurface interface. This interface provides a common framework for performing the most widely used types of analysis such as contouring, feature interpolation, and volumetrics.

Some analysis functions may only be supported by TINs or by rasters, or they may have a unique implementation. If you don't find what you're looking for in ISurface, look to ITinSurface or ISurfaceOp.

ISurface : IFunctionalSurface	*Provides access to members that control surfaces.*
ZFactor: Double	*Multiplication factor applied to all z-values in a TIN to provide unit-congruency between coordinate components.*
AsPolygons (pFeatureClass: IFeatureClass, Type: tagesriSurfaceConversionType, in pClassBreaks: IDoubleArray, in pClassCodes: ILongArray, fieldName: Variant)	*Converts the TIN to a polygon feature class representing slope or aspect.*
Contour (in rootHeight: Double, in interval: Double, pFeatureClass: IFeatureClass, in fieldName: String, in digitsAfterDecimalPoint: Long)	*Interpolates isolines of the TIN surface based on a root value and an interval.*
ContourList (in pBreaks: IDoubleArray, pFeatureClass: IFeatureClass, in fieldName: String, in digitsAfterDecimalPoint: Long)	*The values of isolines, placed into a feature class.*
GetAspectDegrees (in pPoint: IPoint) : Double	*Returns the aspect at the specified location in degrees.*
GetAspectRadians (in pPoint: IPoint) : Double	*Returns the aspect at the specified location in radians.*
GetContour (in pPoint: IPoint, out ppContour: IPolyline, out pElevation: Double)	*Returns a contour passing through the queried point.*
GetElevation (in pPoint: IPoint) : Double	*Returns the z-value of the specified object.*
GetLineOfSight (in pObserver: IPoint, in pTarget: IPoint, out ppObstruction: IPoint, out ppVisibleLines: IPolyline, out ppInvisibleLines: IPolyline, out pbIsVisible: Boolean, in bApplyCurvature: Boolean, in bApplyRefraction: Boolean, pRefractionFactor: Variant)	*Returns a line-of-site indicator interpolated from the TIN based on an input polyline.*
GetProfile (in pShape: IGeometry, out ppProfile: IGeometry, pStepSize: Variant)	*Returns a polyline with z-values interpolated from the TIN.*
GetProjectedArea (in referenceHeight: Double, in Type: tagesriPlaneReferenceType) : Double	*Returns the projected area of the TIN above or below an input z-value.*
GetSlopeDegrees (in pPoint: IPoint) : Double	*Returns the slope at the specified location in degrees.*
GetSlopePercent (in pPoint: IPoint) : Double	*Returns the slope at the specified location in percent.*
GetSlopeRadians (in pPoint: IPoint) : Double	*Returns the slope at the specified location in radians.*
GetSteepestPath (in pPoint: IPoint) : IPolyline	*Returns a polyline interpolated as the steepest path downhill from a specified point.*
GetSurfaceArea (in referenceHeight: Double, in Type: tagesriPlaneReferenceType) : Double	*Returns the TIN's area measured on its surface above or below an input z-value.*
GetVolume (in reference: Double, in Type: tagesriPlaneReferenceType) : Double	*Returns the TIN's volume above or below an input z-value.*
InterpolateShape (in pShape: IGeometry, out ppOutShape: IGeometry, pStepSize: Variant)	*Interpolates z-values for a defined geometric shape.*
InterpolateShapeVertices (in pShape: IGeometry, out ppOutShape: IGeometry)	*Interpolates z-values for a defined geometric shape at its vertices only.*
IsVoidZ (in value: Double) : Boolean	*Returns TRUE if the passed value is equal to the TIN's void value.*
Locate (in pRay: IRay, in hint: Long) : IPoint	*Returns the intersection of the query ray and the displayed feature.*
LocateAll (in pRay: IRay, in hint: Long) : IDoubleArray	*Returns the distances of intersections of the query ray and the displayed feature.*
QueryNormal (in pLocation: IPoint, pNormal: IVector3D)	*Returns the vector normal to the specified triangle.*
QueryPixelBlock (in xOrigin: Double, in yOrigin: Double, in xPixelSize: Double, in yPixelSize: Double, in Type: tagesriRasterizationType, in valueForNoData: Variant, in block: Variant)	*Derives slope, aspect, hillshade, or elevation from the input surface and writes the result to the provided PixelBlock.*
QuerySurfaceLength (in pShape: IGeometry, out pLength: Double, pStepSize: Variant)	*Returns the length of an input polyline measured on the TIN's surface.*

The following code shows how to get the ISurface interface from a TIN:

```
' Get reference to a TIN from a TIN layer.
Dim pTinAdv As ITinAdvanced
Set pTinAdv = pTinLayer.Dataset

' Get reference to ISurface interface.
Dim pSurface As ISurface
Set pSurface = pTinAdv.Surface
```

The next example shows how to get the ISurface interface from a raster:

```
' Get reference to the band collection of a raster.
Dim pRasterBands As IRasterBandCollection
```

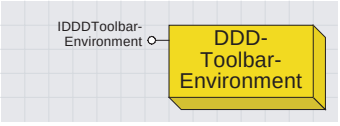
```
Set pRasterBands = pRasterLayer.Raster

' Choose the specific band to use as the surface.
' Most surfaces come from single band rasters. For
' them simply use the first band.
Dim pRasterBand As IRasterBand
Set pRasterBand = pRasterBands.Item(0)

' Create a raster surface object, assign the
' band to it and query for the ISurface interface.
Dim pRasterSurface As IRasterSurface
Set pRasterSurface = New RasterSurface
pRasterSurface.RasterBand = pRasterBand
Set pSurface = pRasterSurface
```

Once you have a handle to the ISurface interface for a TIN or raster, you can use it for both interactive and batch-oriented analysis. ISurface has members useful to both.

The InterpolateShape member can be used for an interactive interpolation tool as can be found on the 3D Toolbar in ArcMap that ships with 3D Analyst. A digitized point, polyline, or polygon can be handed to InterpolateShape to obtain heights. The result is then added back to ArcMap as a graphic or written to the Edit Sketch if a feature class of the appropriate type is being edited. InterpolateShape can also be used to convert an entire 2D feature class into 3D through iterative use of the function on all input features.

```
' This assumes an input polyline has been set
Dim pOutPolyline As IPolyline
pSurface.InterpolateShape pInPolyline, pOutPolyline

If (Not pOutPolyline Is Nothing) Then
  ' Do something with the result.
End If
```

The AsPolygons and Contour members both take feature classes as input arguments. The reason is so one can create an empty feature class of whatever type they choose, be it ArcSDE™, Microsoft® Access, or shapefile, pass it to one of these functions, and have the output sent directly to it. This way neither member needs to be aware of file types or target locations; the output is simply to a feature class. The user must ensure they are empty feature classes of the appropriate type—2D polygons for AsPolygons and 2D polylines for Contour.



The DDDToolbarEnvironment object lets you access the currently selected surface layer.

Both TIN and raster provide low-level access to their data structures. If you can't find an existing function in something like ISurface that does what you need, then you should be able to create it on top of the low-level data access tools.

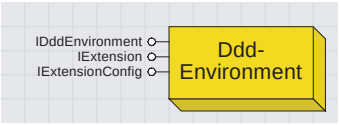
Getting at the current surface on the 3D toolbar

If you want to create tools, such as the contour tool or steepest path tool,

that work on the 3D toolbar on the current surface, you'll need to know how to get at the currently selected layer. The DDDToolbarEnvironment object is used for this purpose.

The following code shows how to get the currently selected layer in the 3D toolbar:

```
' Get the 3D toolbar analysis environment. It's a singleton,
' meaning that only one exists in the application. So when
' you say 'New' you'll just get a reference to the object if
' it already exists.
Dim p3DToolbar As IDDDAnalystToolbarEnvironment
Set p3DToolbar = New DDDAnalystToolbarEnvironment
Dim p3DToolbar As IDDDToolbarEnvironment
Set p3DToolbar = New DDDToolbarEnvironment

' Look to see if there is a layer currently selected in the
' dropdown combo and determine if it's a TIN or raster.
If (Not p3DToolbar.CurrentSelectedLayer Is Nothing) Then
  If (TypeOf p3DToolbar.CurrentSelectedLayer Is ITinLayer) Then
    Dim pTinLayer As ITinLayer
    Set pTinLayer = p3DToolbar.CurrentSelectedLayer
  Else
    Dim pRasterLayer As IRasterLayer
    Set pRasterLayer = p3DToolbar.CurrentSelectedLayer
  End If
End If
```



The DDDEnvironment object is used for managing licensing of the 3D Analyst extension.

3D Analyst is a licensed extension. Its functions make calls to a license manager to determine if a license has been reserved. Custom tools using licensed 3D functionality that operate exclusively in ArcScene need not worry about licensing as ArcScene handles this when it starts up. Tools in other environments, such as ArcMap, ArcCatalog, or custom stand alone applications, need to do some work though.

IDddEnvironment : IUnknown	***Provides access to the 3D Analyst license.***
CanIRun (in suppressMessages: Boolean) : Boolean	*Indicates if 3D license is provided.*
LicenseIsAvailable: Boolean	*Indicates if 3D license is available.*
EndRun	*Gives back the license.*
GetRasterSettings: IUnknown Pointer	*Returns global raster settings.*

Both ArcMap and ArcCatalog support extensions and load them if they're installed. In order for your tool to find out if a license is available (i.e., as a prerequisite for enabling it), it needs to ask the 3D extension object.

The following code is used to determine if a 3D license has been reserved:

```
' Get the IExtensionConfig interface from the 3D extension object
Dim pExtConfig As IExtensionConfig
Set pExtConfig = Application.FindExtensionByName("3D Analyst")

' Indicate whether a license has been checked out.
MsgBox "3D extension is enabled: " & (pExtConfig.State =
esriESEnabled)
```

In a custom standalone application you'll need to create an extension manager, have it create a 3D extension, and reserve a 3D license. The extension manager needs to live at least as long as licenses are required. Frequently, this is the duration of the application.

This example can be used in a custom standalone application to acquire a 3D license:

```
' Get the ProgId of the 3D extension.
Dim puid As New UID
puid.Value = "esricore.DDDEnvironment"

' Create a extension manager and have it create
' a 3D extension object. The extension manager
' should live as long as the application, or
' at least as long as licenses are required.
Dim m_pExtAdmin As IExtensionManagerAdmin ' should be module level
variable
Set m_pExtAdmin = New ExtensionManager
m_pExtAdmin.AddExtension puid, 0
```

```
' Get the 3D extension from the manager.
Dim pExtManager As IExtensionManager
Set pExtManager = m_pExtAdmin
Dim pExtConfig As IExtensionConfig
Set pExtConfig = pExtManager.FindExtension("3D Analyst")

' See if a license is potentially available and if so
' try to get one.
If (Not pExtConfig.State = esriESUnavailable) Then
  pExtConfig.State = esriESEnabled
  MsgBox "3D license checked out: " & _
    (pExtConfig.State = esriESEnabled)
Else
  MsgBox "No 3D licenses exist on this system."
End If
```