CLIENT GRAPHICS

IN

INVENTOR

Table of Contents

[Client Graphics 3](#_Toc276047113)

[API Object Model 4](#_Toc276047114)

[Types of Client Graphics 4](#_Toc276047115)

[Regular Client Graphics 4](#_Toc276047116)

[Interaction Client Graphics 4](#_Toc276047117)

[Graphics Data 5](#_Toc276047118)

[Data Objects 5](#_Toc276047119)

[Subsidiary Objects 5](#_Toc276047120)

[General Procedure 6](#_Toc276047121)

[Graphic Primitives 7](#_Toc276047122)

[Point 7](#_Toc276047123)

[Line 7](#_Toc276047124)

[Line Strip 7](#_Toc276047125)

[Triangle 8](#_Toc276047126)

[Triangle Strip 8](#_Toc276047127)

[Triangle Fan 9](#_Toc276047128)

[Text 10](#_Toc276047129)

[Curve 10](#_Toc276047130)

[Surface 11](#_Toc276047131)

[Using Index Sets 12](#_Toc276047132)

[Using ‘Strip’ graphics 13](#_Toc276047133)

[Using Model Native Data 14](#_Toc276047134)

[Basic Properties 15](#_Toc276047135)

[Interaction Client Graphics 16](#_Toc276047136)

[Store Client Graphics 17](#_Toc276047137)

[Advanced Functionality 18](#_Toc276047138)

[Slice 18](#_Toc276047139)

[Texture mapping 18](#_Toc276047140)

[Further References 18](#_Toc276047141)

[Sample code 18](#_Toc276047142)

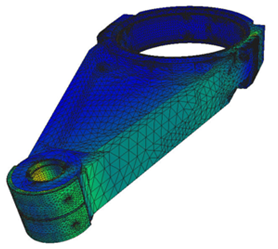
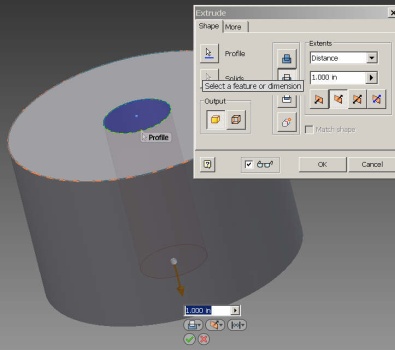
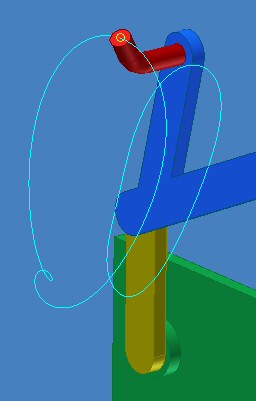
## Client Graphics

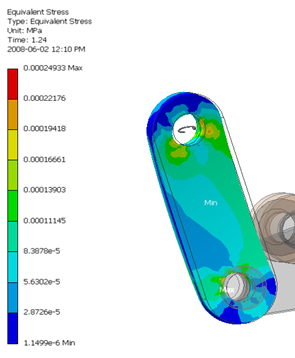
Autodesk Inventor’s custom (client) graphics provide the ability to draw basic graphics alongside Inventor’s native graphics via the Application programmers Interface (API). Client Graphics are commonly used Inventor ‘add-in’ applications to represent their custom objects, to create interactive previews during the add-in commands, displaying the results of various analyses and for creating custom manipulators within a command. An example is a milling application in CAM, where the toolpath and the tool can be shown using client graphics. Another example is displaying the results of a finite element analysis performed on the Inventor model.

Inventor provides an API for programming such transient graphics independently of the underlying graphics platform that Inventor is running on, which provides major benefits in terms of portability and managing graphics library changes.

The Inventor API provides access to its own set of graphic primitives. These primitives include points, lines, triangles, text, and are collectively referred to as **Client Graphics**.

Client Graphics allow the developer to provide visual cues to actions being performed, and Inventor uses this feature itself in many scenarios, e.g. when the user creates an extrusion, a preview of the extruded part is displayed. While defining the extrude feature, a visual cue indicates what the final result will be. An add-in developer can provide similar visual cues within their own application.





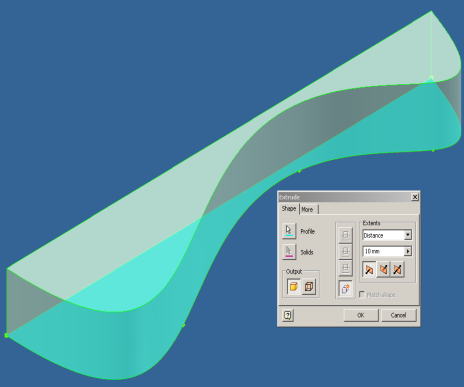


Fig 1 : applications with client graphics

## API Object Model

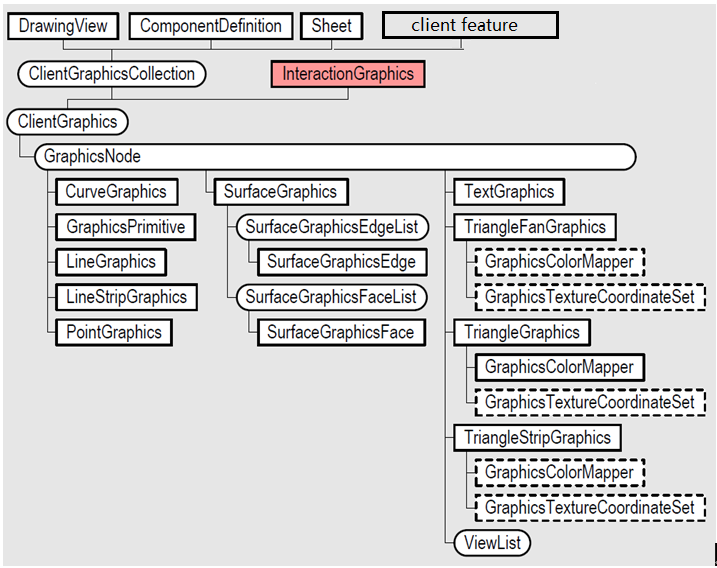


Fig 2 : API Model Object

The diagram above outlines the API Object model associated with Inventor’s client graphics.

These are the major objects in the model:-

* **ClientGraphics**: Owner of a group of graphics. Control the visibility and selectability of the group.
* **GraphicsNode**: Owner of graphics primitives. Control the visibility, selectability, color, render style, position and transformation.
* **Graphic primitives (TextGraphics, TriangleGraphics etc)**: The graphics displayed in the Inventor graphics window: points, lines, triangles, curves, text, and surface.

## Types of Client Graphics

Regular Client Graphics

These types of client graphics are associated with an Inventor document. They are transient unless associated with a client feature, which means they are not retained by the document when the document is closed. One document, sheet or drawingview can have more than one client graphics object associated with it. Each client graphics object can have many graphics nodes. Each node can contain any number and any type of graphic **primitives**.

Interaction Client Graphics

This type of client graphics is available when using Interaction Events. They are always transient. The InteractionEvents object can have any number of nodes and graphics primitives, like **Regular Client Graphics**.

## Graphics Data

In Inventor most client graphics are defined using two different types of objects; graphics data and graphic primitives. The graphics data define the low-level information that can be used to define primitives. By separating the data from the primitives, the data can be re-used by many different graphics primitives.

#### Data Objects

For **Point, Line, LineStrip, Triangle, TriangleStrip,** and **TriangleFans**, the data object provides the list of coordinates that the graphics can be based upon, along with any other necessary data sets such as the ‘color’ and ‘surface normal vector’ information.

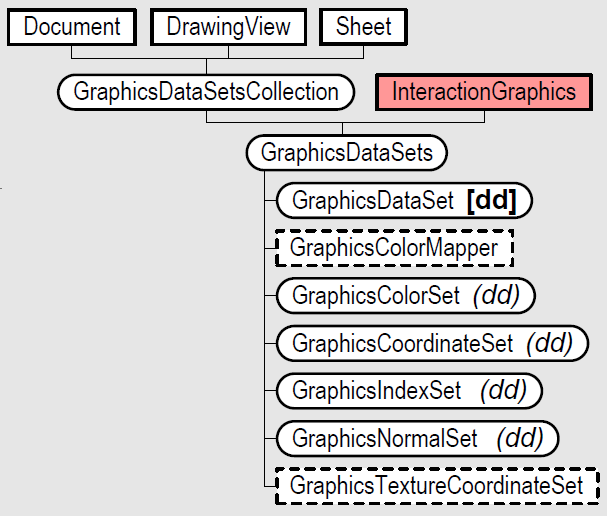


Fig 3 : Data Objects

* **Coordinate Sets**: The coordinates used to specify the vertices for the graphic sets.
* **Color Sets**: A set of colors used by a graphic set. It overrides any other color information assigned to the set. Colors in the color set can be bound to the entire graphic set, each individual primitive in the set (e.g. each triangle in a triangle strip), or each vertex in the set (i.e. interpolated color).
* **Normal Sets**: Contains a set of normal vectors (which can be used to define how the lighting is calculated for the triangles).
* **Index Sets**: Each graphics primitive can access this set to more efficiently use an associated coordinate or color.

#### Subsidiary Objects

* **TransientGeometry**

The TransientGeoemtry object is a utility object you use to create various points, curves, surfaces, and mathematical objects like vectors, and matrices. In association with client graphics, the ability to create different types of curves is useful because you can use these as input to define **Curve** type graphics.

* **TransientBrep**

The TransientBRep object is used to create transient surface and solid models. These can be used to create **Surface** graphics.

## General Procedure

Creation of Client Graphics follows the general procedure:

1. Get the GraphicsDataSetsCollection object from the Document
2. Add a GraphicsDataSets object and create the graphics data

(for Curve or Surface, create graphics data using the TransientGeometry and TransientBrep objects)

1. Get the ClientGraphicsCollection from the graphics owner, which can be:-

* A component in Part or Assembly
* A View or a Sheet in Drawing document
* InteractionEvents for Interaction Client Graphics
* Client Feature

1. Add a ClientGraphics object to ClientGraphicsCollection
2. Add as many GraphicsNodes as you need
3. Add GraphicsPrimitive objects to the node(s)
4. Assign graphics data to the primitives
5. Update the view(s)

**‘Draw LineGraphics (two points)**

Public Sub DrawLineGraphics()

Dim oDoc As Document

Set oDoc = ThisApplication.ActiveDocument

'Get the GraphicsDataSetsCollection object from the Document. Add a GraphicsDataSets object

Dim oDataSets As GraphicsDataSets

Set oDataSets = oDoc.GraphicsDataSetsCollection.Add("TestCG")

Dim oCompDef As ComponentDefinition

Set oCompDef = oDoc.ComponentDefinition

'Get the ClientGraphicsCollection from the graphics owner. Add a ClientGraphics

Dim oClientGraphics As ClientGraphics

Set oClientGraphics = oCompDef.ClientGraphicsCollection.Add("TestCG")

Dim oCoordSet As GraphicsCoordinateSet

Set oCoordSet = oDataSets.CreateCoordinateSet(1)

' Create graphics data. In this case, prepare two points for line graphics

Dim oPointCoords(5) As Double

oPointCoords(0) = 0: oPointCoords(1) = 0: oPointCoords(2) = 0 ‘point 1

oPointCoords(3) = 1: oPointCoords(4) = 0: oPointCoords(5) = 0 ‘point 2

Call oCoordSet.PutCoordinates(oPointCoords)

'Add GraphicsNodes

Dim oGraphicsNode As GraphicsNode

Set oGraphicsNode = oClientGraphics.AddNode(1)

'Add GraphicsPrimitive to node

Dim oGraphic As LineGraphics

Set oGraphic = oGraphicsNode.AddLineGraphics

'Assign graphics data to the primitives

oGraphic.CoordinateSet = oCoordSet

'update the view

ThisApplication.ActiveView.Update

End Sub

## Graphic Primitives

Point

Point is the simplest graphics object available. You could change the point style such as circular dot, a cross etc, and from Inventor 2011, a Point can be displayed with an **image**.

<code snippet : **Point**>

Dim oPointCoords(8) As Double

oPointCoords(0) = 0: oPointCoords(1) = 0: oPointCoords(2) = 0

oPointCoords(3) = 0: oPointCoords(4) = 1: oPointCoords(5) = 0

oPointCoords(6) = 1: oPointCoords(7) = 1: oPointCoords(8) = 1

' Create an image set

Dim oImageSet As GraphicsImageSet

Set oImageSet = oDataSets.CreateImageSet(oDataSets.count + 1)

Dim oImage As IPictureDisp

Set oImage = LoadPicture("C:\Temp\MyImage.bmp")

Call oImageSet.Add(1, oImage)

‘add point graphics and assign the data

Call oCoordSet.PutCoordinates(oPointCoords)

Dim oPointGraphics As **PointGraphics**

Set oPointGraphics = oPointNode.AddPointGraphics

oPointGraphics.PointRenderStyle = kFilledCrossPointStyle

oPointGraphics.CoordinateSet = oCoordSet

oPointGraphics.BurnThrough = True

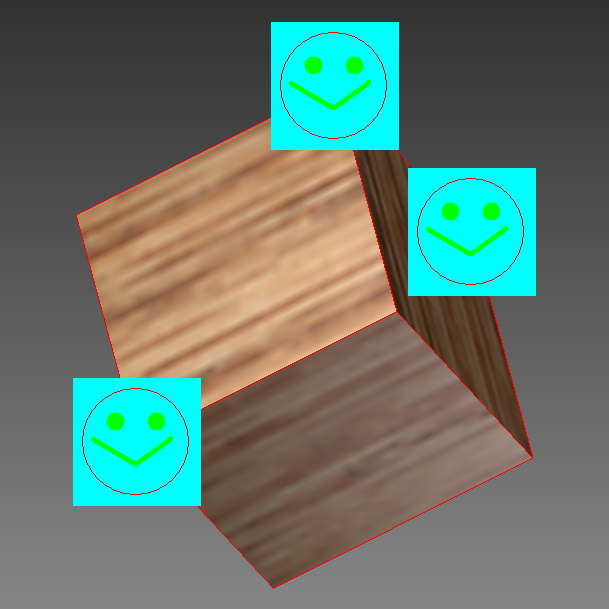


Fig 4: Point Graphics

Line

These are individual line segments. [Line](mk:@MSITStore:C:\Program%20Files\Autodesk\Inventor%202011\Help\admapi_15_0.chm::/Inventor__Line.html) graphics use two coordinates to define a line, and then the next two coordinates to define the next line, and so on through the defined coordinates.

Line Strip

These are a connected set of lines. [Line](mk:@MSITStore:C:\Program%20Files\Autodesk\Inventor%202011\Help\admapi_15_0.chm::/Inventor__Line.html) strips use the first two coordinates to define the first line and then the last point of the first line becomes the first point of the second line and the next coordinate is used as the end point of the second line. This results in the set of points being connected by a continuous set of lines, drawing a continuous curve. So to define 2 connected lines, just 3 coordinates would be required.

<code snippet: Line or LineStrip>

Dim oPointCoords(11) As Double ‘create 4 points

oPointCoords(0) = 0: oPointCoords(1) = 0: oPointCoords(2) = 0

oPointCoords(3) = 1: oPointCoords(4) = 1: oPointCoords(5) = 0

oPointCoords(6) = 2: oPointCoords(7) = 0: oPointCoords(8) = 0

oPointCoords(9) = 3: oPointCoords(10) = 1: oPointCoords(11) = 0

Call oCoordSet.PutCoordinates(oPointCoords)

‘Line: every 2 points for 1 Line

Dim oGraphic As **LineGraphics**

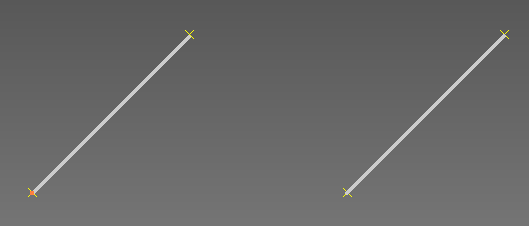
Set oGraphic = oGraphicsNode.AddLineGraphics

‘Line strip: totally, 3 lines

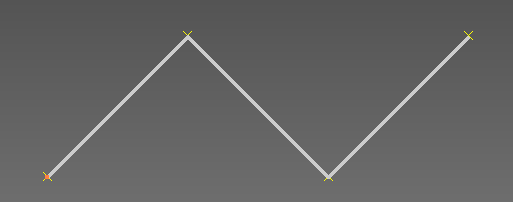
‘Dim oGraphic As **LineStripGraphics**

‘Set oGraphic = oGraphicsNode.AddLineStripGraphics

oGraphic.CoordinateSet = oCoordSet



**Fig 5: Line Graphics**



**Fig 6: Line Strip Graphics**

Triangle

These are used to define individual triangles. Each set of three coordinates defines a triangle. The next tree coordinates then define the next triangle, and so on through all of the coordinates that are provided.

Triangle Strip

These provide a connected set of triangles. The first three coordinates define a triangle and the next coordinate defines another triangle using the previous two coordinates. You could also specify the strip length. This is provided for performance reasons.

<code snippet: Triangle or Triangle Strip>

Dim oPointCoords(20) As Double ‘create 7 points,

oPointCoords(0) = 0: oPointCoords(1) = 0: oPointCoords(2) = 0

oPointCoords(3) = 1: oPointCoords(4) = 1: oPointCoords(5) = 0

oPointCoords(6) = 2: oPointCoords(7) = 0: oPointCoords(8) = 0

oPointCoords(9) = 3: oPointCoords(10) = 1: oPointCoords(11) = 0

oPointCoords(12) = 4: oPointCoords(13) = 0: oPointCoords(14) = 0

oPointCoords(15) = 5: oPointCoords(16) = 1: oPointCoords(17) = 0

oPointCoords(15) = 5: oPointCoords(16) = 1: oPointCoords(17) = 0

oPointCoords(18) = 6: oPointCoords(19) = 0: oPointCoords(20) = 0

Call oCoordSet.PutCoordinates(oPointCoords)

‘Triangle: every 3 points for 1 triangle

Dim oGraphic As **TriangleGraphics**

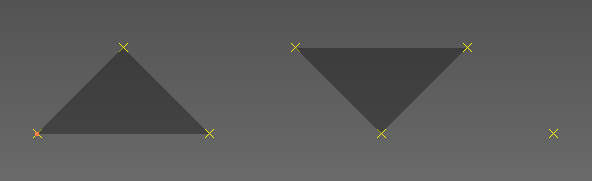
Set oGraphic = oGraphicsNode.AddTriangleGraphics

‘or Triangle Strip: totally 5 triangles

‘Dim oGraphic As **TriangleStripGraphics**

‘Set oGraphic = oGraphicsNode.AddTriangleStripGraphics

oGraphic.CoordinateSet = oCoordSet



**Fig 7: Triangle**

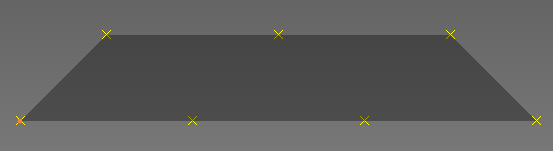


Fig 8: Triangle Strip

Triangle Fan

This defines a set of connected triangles. The first coordinate is shared in all triangles. The first three coordinates define a triangle. And the next coordinate defines another triangle using the previous coordinates: one of which is the first coordinate. If all the coordinates surround the first point, then the resulting graphics looks like a fan. Again this facility is provided primarily for performance reasons.

<code snippet: TriangleFan>

Dim oPointCoords(17) As Double ‘Create 6 points

oPointCoords(0) = 0: oPointCoords(1) = 0: oPointCoords(2) = 0

oPointCoords(3) = 1: oPointCoords(4) = 0: oPointCoords(5) = 0

oPointCoords(6) = 0: oPointCoords(7) = 1: oPointCoords(8) = 0

oPointCoords(9) = -1: oPointCoords(10) = 0: oPointCoords(11) = 0

oPointCoords(12) = 0: oPointCoords(13) = -1: oPointCoords(14) = 0

oPointCoords(15) = 1: oPointCoords(16) = 0: oPointCoords(17) = 0

‘totally 4 triangles. All around the first point

Dim oGraphic As **TriangleFanGraphics**

Set oGraphic = oGraphicsNode.AddTriangleFanGraphics

oGraphic.CoordinateSet = oCoordSet

Text

TransientGeometry creates the anchor point for **TextGraphics**. Other properties are: text string, font, bold, horizontal alignment etc.

<code snippet: Text>

Dim oTG As TransientGeometry

Set oTG = ThisApplication.TransientGeometry

Dim oTextGraphics As **TextGraphics**

Set oTextGraphics = oNode.AddTextGraphics

oTextGraphics.Text = Text

oTextGraphics.Anchor = oPosition

oTextGraphics.Bold = True

oTextGraphics.Font = "Arial"

oTextGraphics.FontSize = 40

oTextGraphics.HorizontalAlignment = kAlignTextLeft

oTextGraphics.Italic = True

Call oTextGraphics.PutTextColor(0, 255, 0)

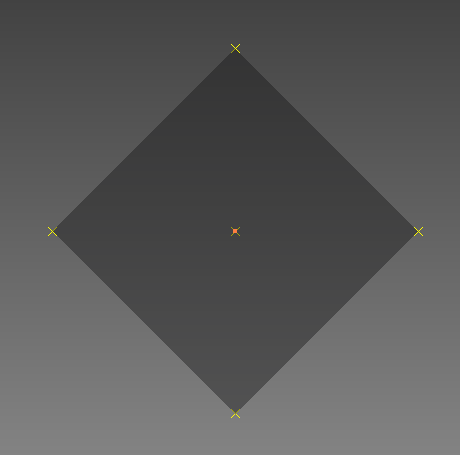
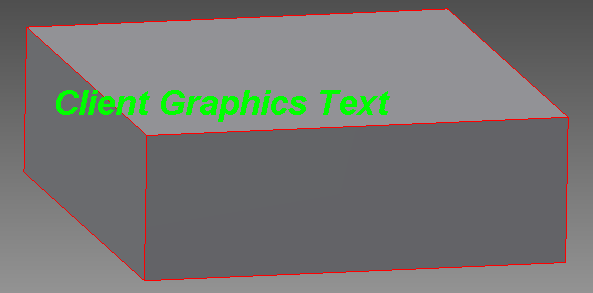
 

Fig 9: Triangle Fan Fig10: Text

Curve

Curves include the following objects: LineSegment, Circle, Arc3d, EllipseFull, EllipticalArc and BSplineCurve. The object TransientGeometry is used to create the geometry and client graphics then uses this geometry as input. You can also use the geometry from a model’s native data too.

<code snippet: Curve>

Dim oTG As TransientGeometry

Set oTG = ThisApplication.TransientGeometry

Dim oCircle As Inventor.Circle

Set oCircle = oTG.CreateCircle(oTG.CreatePoint(0, 0, 0),

oTG.CreateUnitVector(0, 0, 1), 5#)

Dim oGraphic As **CurveGraphics**

Set oGraphic = oGraphicsNode.AddCurveGraphics(oCircle)

Surface

TransientBrep is used to create the underlying geometry data, and client graphicsobjects then use this geometry. Again you can use the model’s native data: face, faces or face collection too.

<code snippet: Surface>

Dim oTransientBRep As **TransientBRep**

Set oTransientBRep = ThisApplication.TransientBRep

Dim oTG As TransientGeometry

Set oTG = ThisApplication.TransientGeometry

Dim oBody As SurfaceBody

Set oBody = oTransientBRep.CreateSolidCylinderCone(

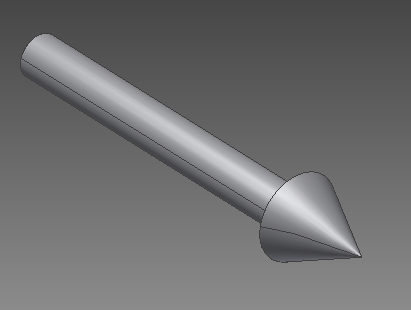
oTG.CreatePoint(0, 0, 0),

oTG.CreatePoint(0, 10, 0

5, 5, 0)

Dim oSurfaceGraphics As **SurfaceGraphics**

Set oSurfaceGraphics = oSurfacesNode.AddSurfaceGraphics(oBody)



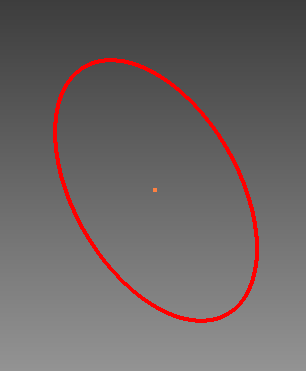


Fig 11: Curve Fig 12: Surface

## Using Index Sets

The Index Set provides a flexible way to arrange the coordinate set, color set and normal set (for triangles). For example, with the same coordinate data, the figure below, on the left, creates the line graphics from 1-2-3-4 where all lines have the same color, while the figure on the right creates the line from 1-3-2-4 and each line has a different color defined using an index set.

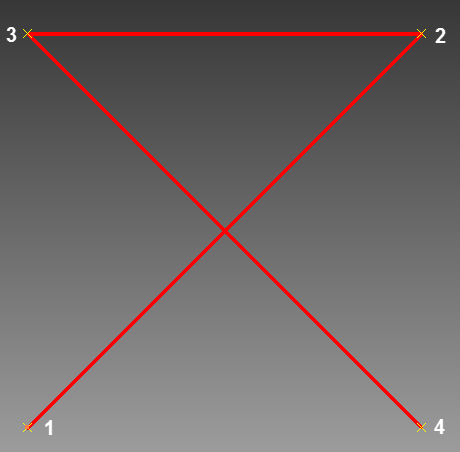
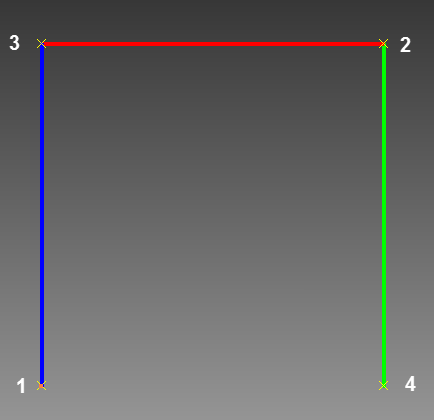


Fig 13: Line graphic without index

Fig 14: Line graphics with color and

coordinate index

<code snippet: Line, With Coordinate Index Set and Color Set>

Dim oPointCoords(11) As Double 'create 4 points

oPointCoords(0) = 0: oPointCoords(1) = 0: oPointCoords(2) = 0

oPointCoords(3) = 1: oPointCoords(4) = 1: oPointCoords(5) = 0

oPointCoords(6) = 0: oPointCoords(7) = 1: oPointCoords(8) = 0

oPointCoords(9) = 1: oPointCoords(10) = 0: oPointCoords(11) = 0

Call oCoordSet.PutCoordinates(oPointCoords)

'Line strip: totally, 3 lines

Dim oGraphic As LineGraphics

Set oGraphic = oGraphicsNode.AddLineGraphics

oGraphic.LineWeight = 5

‘Create Coordinate Index Set

Dim oCoordinateIndex As **GraphicsIndexSet**

Set oCoordinateIndex = oDataSets.CreateIndexSet(oDataSets.count + 1)

oCoordinateIndex.Add 1, 1 'from point 1

oCoordinateIndex.Add 2, 3 'connect to point 3

oCoordinateIndex.Add 3, 3 'from point 3

oCoordinateIndex.Add 4, 2 'connect to point 2

oCoordinateIndex.Add 5, 2 'from point 2

oCoordinateIndex.Add 6, 4 'connect to point 4

'Create the color set: two colors.

Dim oColorSet As **GraphicsColorSet**

Set oColorSet = oDataSets.CreateColorSet(oDataSets.count + 1)

Call oColorSet.Add(1, 255, 0, 0)

Call oColorSet.Add(2, 0, 255, 0)

' Create the index set for color

Dim oColorIndex As **GraphicsIndexSet**

Set oColorIndex = oDataSets.CreateIndexSet(oDataSets.count + 1)

oColorIndex.Add 1, 2 'line 1 uses color 2

oColorIndex.Add 2, 1 'line 2 uses color 1

oColorIndex.Add 3, 2 'line 3 uses color 2

oGraphic.CoordinateSet = oCoordSet

oGraphic.CoordinateIndexSet = oCoordinateIndex

oGraphic.ColorIndexSet = oColorIndex

oGraphic.ColorSet = oColorSet

oGraphic.ColorBinding = **kPerItemColors**

## Using ‘Strip’ graphics

Strip Graphics are used to group the coordinates associated with a series of connected graphics objects and are only available with LineStrip, TriangleStrip or TriangleFan. You can specify the strip length. e.g. in the code below, points 1,2,3 are defined as one triangle strip, whilst points 4,5,6,7 are used to defined as another strip. Each strip is independent of the other strips. You can also specify the color for each.

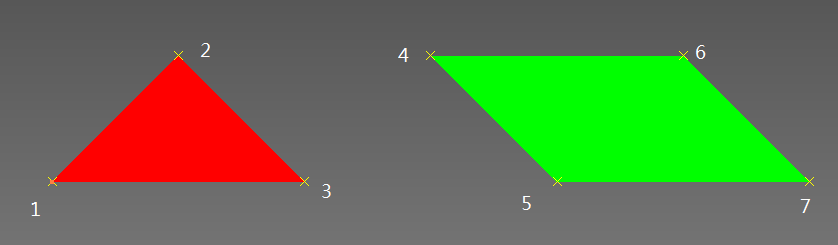


Fig 15: Triangle using strip

<code snippet: Triangle Strip, Set Strip>

Dim oPointCoords(20) As Double ‘create 7 points,

oPointCoords(0) = 0: oPointCoords(1) = 0: oPointCoords(2) = 0

oPointCoords(3) = 1: oPointCoords(4) = 1: oPointCoords(5) = 0

oPointCoords(6) = 2: oPointCoords(7) = 0: oPointCoords(8) = 0

oPointCoords(9) = 3: oPointCoords(10) = 1: oPointCoords(11) = 0

oPointCoords(12) = 4: oPointCoords(13) = 0: oPointCoords(14) = 0

oPointCoords(15) = 5: oPointCoords(16) = 1: oPointCoords(17) = 0

oPointCoords(15) = 5: oPointCoords(16) = 1: oPointCoords(17) = 0

oPointCoords(18) = 6: oPointCoords(19) = 0: oPointCoords(20) = 0

Call oCoordSet.PutCoordinates(oPointCoords)

‘or Triangle Strip: totally 5 triangles

Dim oGraphic As **TriangleStripGraphics**

Set oGraphic = oGraphicsNode.AddTriangleStripGraphics

Dim oStrip(1) As Long

oStrip(0) = 3 ‘ point 1,2,3 for strip 1

oStrip(1) = 4 ‘ point 4,5,6,7 for strip 2

**oGraphic.PutStripLengths oStrip**

'Create the color set: two colors.

Dim oColorIndex As GraphicsIndexSet

Set oColorIndex = oDataSets.CreateIndexSet(oDataSets.count + 1)

oColorIndex.Add 1, 1

oColorIndex.Add 2, 2

oColorIndex.Add 3, 1

oColorIndex.Add 4, 2

oColorIndex.Add 5, 1

' Create the index set for color

Dim oColorSet As GraphicsColorSet

Set oColorSet = oDataSets.CreateColorSet(oDataSets.count + 1)

Call oColorSet.Add(1, 255, 0, 0)

Call oColorSet.Add(2, 0, 255, 0)

oGraphic.CoordinateSet = oCoordSet

oGraphic.ColorIndexSet = oColorIndex

oGraphic.ColorSet = oColorSet

oGraphic.ColorBinding = **kPerStripColors**

## Using Model Native Data

Most scenarios where Client Graphics are required are based on the models native geometry. It is quite easy to get the data using Inventor API objects. e.g. SurfaceBody, Face, Edge, Vertex in Part or Assembly Documents, and DrawingCurve in drawing documents. The code below creates and transforms a surface using an existing SurfaceBody from the model.

It is also possible to obtain a coordinate data set from the existing SurfaceBody, Edge or Face objects using the CalculateStrokes method. You determine the tolerance (the number of points you need) by first calling **GetExistingStrokeTolerances**. You may need to do this multiple times in order to assess the average tolerance for the model. Similarly, obtain triangular facets from A SurfaceBody or Face using **CalculateFacets**.

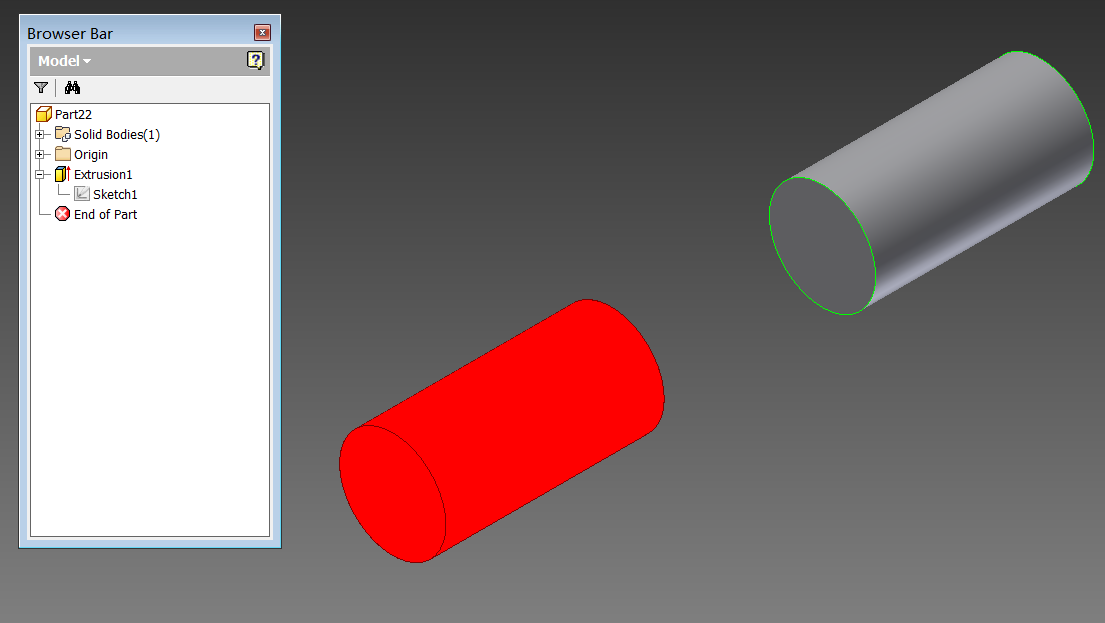


Fig 16: Using surface body

<code snippet: Using model native data>

Dim oPartDef As PartComponentDefinition

Set oPartDef = ThisApplication.ActiveDocument.ComponentDefinition

Dim oTransientBRep As TransientBRep

Set oTransientBRep = ThisApplication.TransientBRep

‘by Transient Brep class to copy the surface body

Dim oBody As SurfaceBody

Set oBody = oTransientBRep.Copy(oCompDef.SurfaceBodies.Item(1))

' Create client graphics based on the transient body

Dim oSurfaceGraphics As SurfaceGraphics

Set oSurfaceGraphics = oSurfacesNode.AddSurfaceGraphics(oBody)

‘move the graphics to a new location

Dim oMatrix As Matrix

Set oMatrix = ThisApplication.TransientGeometry.CreateMatrix

oMatrix.SetTranslation ThisApplication.TransientGeometry.CreateVector(0, 0, 100)

oSurfacesNode.Transformation = oMatrix

## Basic Properties

Graphics has some properties to control the basic behavior.

**Render Style**

By default, the linetype, lineweight and color will use the settings on RenderStyle until they are overridden by **LineType, LineWeight, Color** (for Curve and Surface objects), and **ColorSet** (for Line, Triangle and TriangleFan objects).

**Point Style**

Specific to Point objects only, and controls the point style.

**Visibility**

Controls whether or not the [graphics](mk:@MSITStore:C:\Program%20Files\Autodesk\Inventor%202011\Help\admapi_15_0.chm::/Inventor__GraphicsNode.html) are visible.

**Selectable**

Specifies whether client graphics can be selected when the Select command is running.

**Burn Through**

This specifies whether or not graphics are always visible even if they are blocked by other objects.

**Transformation**

This property is used to transform a graphics node.

**Transform Behavior**

Two special transform behaviors: front facing and pixel scaling.

* **Front Facing**

Client graphics with this property do not rotate as the view is rotated but maintain the same orientation on the screen. They are positioned at a specified location within model space and their position on the screen will change as the view is zoomed in and out and scrolled, but their orientation will not change.

* **Pixel Scaling**

Here client graphics maintain the same size and position relative to the screen. As the user zooms in and out in the graphic window, these objects remain static.

Any graphics object can also have no transform behavior (which means it's size, position, and orientation are maintained relative to model space), front facing, pixel scaling, or front facing & pixel scaling. By default an object has no transform behavior, with the exception of text. **Text** always has front facing behavior regardless of the behavior type set through this method.

**Anchor Behavior**

This will anchor the primitive in view space so that the graphics will display in one fixed position, no matter how the view changes.

## Interaction Client Graphics

Interaction Client Graphics operate in a similar manner to regular ClientGraphics, except that it all happens in the context of InteractionEvents. ClientGraphics created via InteractionGraphics are optimized for this environment , and run much faster. They are well suited to real-time feedback during a command. As the client graphics are displayed in the context of an InteractionEvents object, they can take advantage of mouse movement information, selection information, in fact any events supported by the InteractionEvents objects. These graphics are automatically removed once the associated InteractionEvents object stops.

The interaction graphics can be **Preview** or **Overlay**. Preview graphics are equivalent to regular Client Graphics. The only difference between Overlay and Preview graphics is that Overlay graphics can be drawn independently, in a single view (window), and then merged with the last full redraw of the scene by drawing the graphics in a special "overlay" plane. It doesn't mean that the graphics are always drawn on top. You can position them anywhere you want and you can toggle the "burn through" option. In short, anything you can do with preview graphics you can do with overlay graphics.

The creation of interaction graphics is not transacted. This is done for performance reasons, and to avoid some side effects, like clearing of the selection set. This also means that transacting operations cannot be mixed with preview graphics.

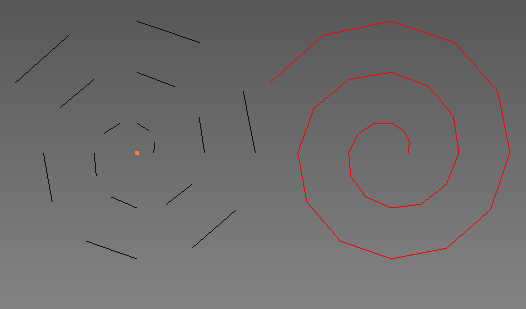


Fig 17: Interaction graphics – removed when event stops

<code snippet: interaction events (Overlay)>

‘start interaction event and get the overlay graphics

Dim oIE As InteractionEvents

Set oIE = ThisApplication.CommandManager.CreateInteractionEvents

oIE.Start

Dim oClientGraphics As ClientGraphics

Set oClientGraphics = oIG.OverlayClientGraphics

‘create line strip graphics

Dim oLineStrip As LineStripGraphics

Set oLineStrip = oLineStripNode.AddLineStripGraphics

' Assign the same coordinate set to the line strip.

oLineStrip.CoordinateSet = oCoordSet

' Update the view to see the resulting spiral.

oIG.UpdateOverlayGraphics ThisApplication.ActiveView

## Store Client Graphics

In general, ClientGraphics, will be maintained and transformed by Inventor for the duration of the session only. Using the **GraphicsDataSetsCollection.Add2** method will allow the data to be saved with the file. When the document is opened, the ClientGraphics are persisted, and are still available to be further manipulated.

In Part or Assembly documents, you can attach client graphics to **ClientFeature** objects, so the graphics will be stored with document. The presumption is the graphics data is created by **GraphicsDataSetsCollection.Add2**. In Drawing documents only the data can be saved. A developer’s application has to redraw the graphics using the client graphics data after the Drawing document has been re-opened.

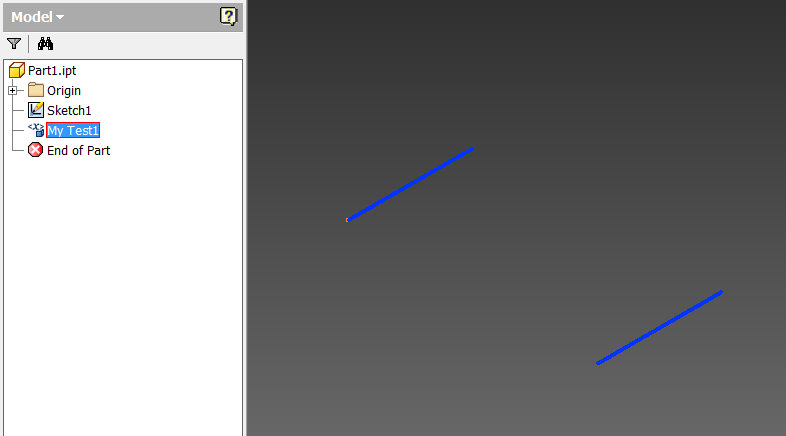


Fig 18 : Client Feature with Client Graphics

<code snippet: store client graphics with client feature>

Dim invCF As ClientFeature

‘get/create ClientFeature

Dim invCFD As ClientFeatureDefinition

invCFD = invCF.Definition

‘Create & store graphics data

Dim oGraphicsData As GraphicsDataSets

Set oGraphicsData = oDoc.GraphicsDataSetsCollection.**Add2**("TestCG\_StoreData", True)

‘create graphics in the ClientFeature

Dim oClientGraphics As Inventor.ClientGraphics

Set oClientGraphics = invCFD.ClientGraphicsCollection("ClientFeatureTest")

‘create graphics node

Dim oGraphicsNode As GraphicsNode

Set oGraphicsNode = oClientGraphics.AddNode(oClientGraphics.count + 1)

Dim oPointCoords(11) As Double 'create 4 points

oPointCoords(0) = 0: oPointCoords(1) = 0: oPointCoords(2) = 0

oPointCoords(3) = 1: oPointCoords(4) = 1: oPointCoords(5) = 0

oPointCoords(6) = 2: oPointCoords(7) = 0: oPointCoords(8) = 0

oPointCoords(9) = 3: oPointCoords(10) = 1: oPointCoords(11) = 0

Call oCoordSet.PutCoordinates(oPointCoords)

'Lines: totally, 2 lines

Dim oGraphic As LineGraphics

Set oGraphic = oGraphicsNode.AddLineGraphics

oGraphic.CoordinateSet = oCoordSet

## Advanced Functionality

#### Slice

This provides the functionality that slices the graphics based on the input planes and optionally caps the sliced end.

#### Texture mapping

Texture mapping provides the ability to more accurately define color mapping to the model. For example, with the previous functionality you were limited to assigning colors to vertices and then relying on Inventor to interpolate the color across the triangle. If you had a plane constructed of two triangles you would not be able to define any intermediate colors between the vertices.

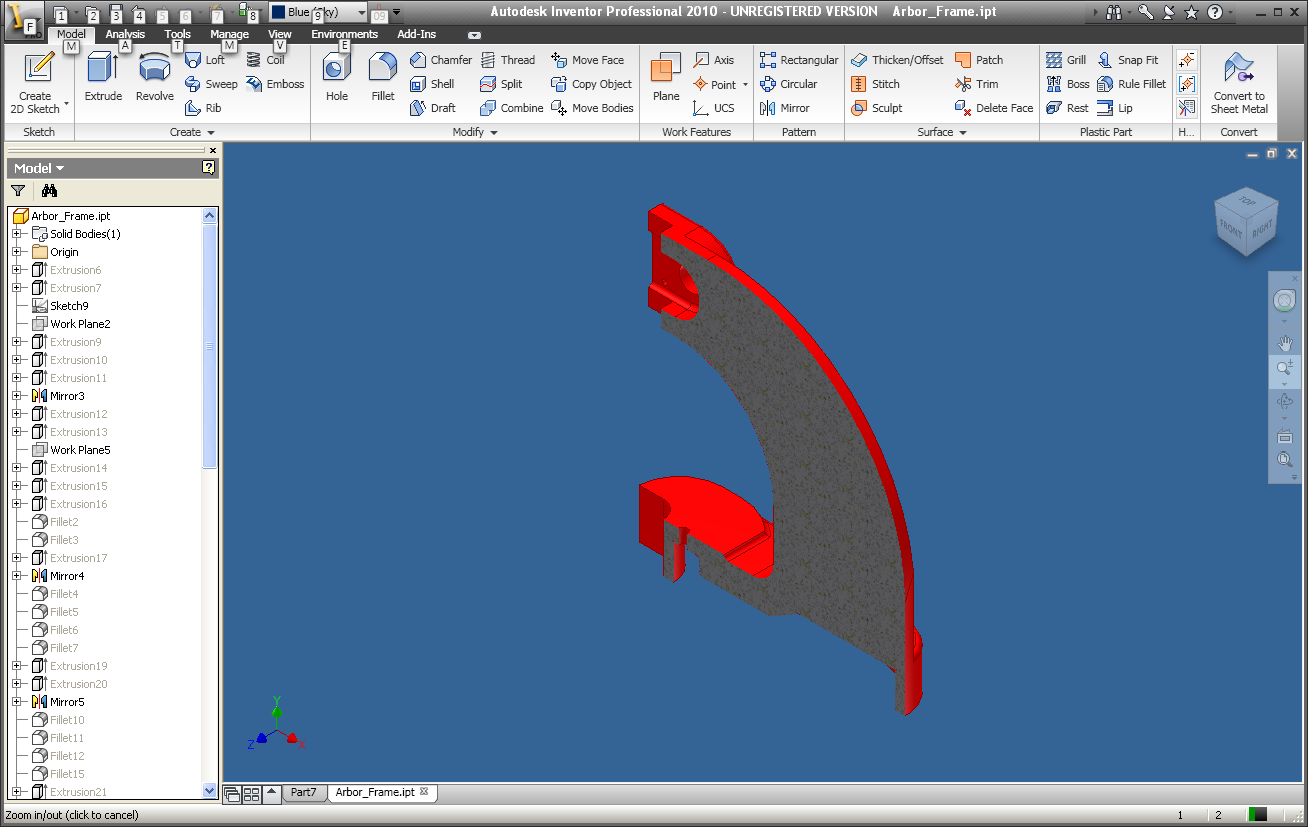
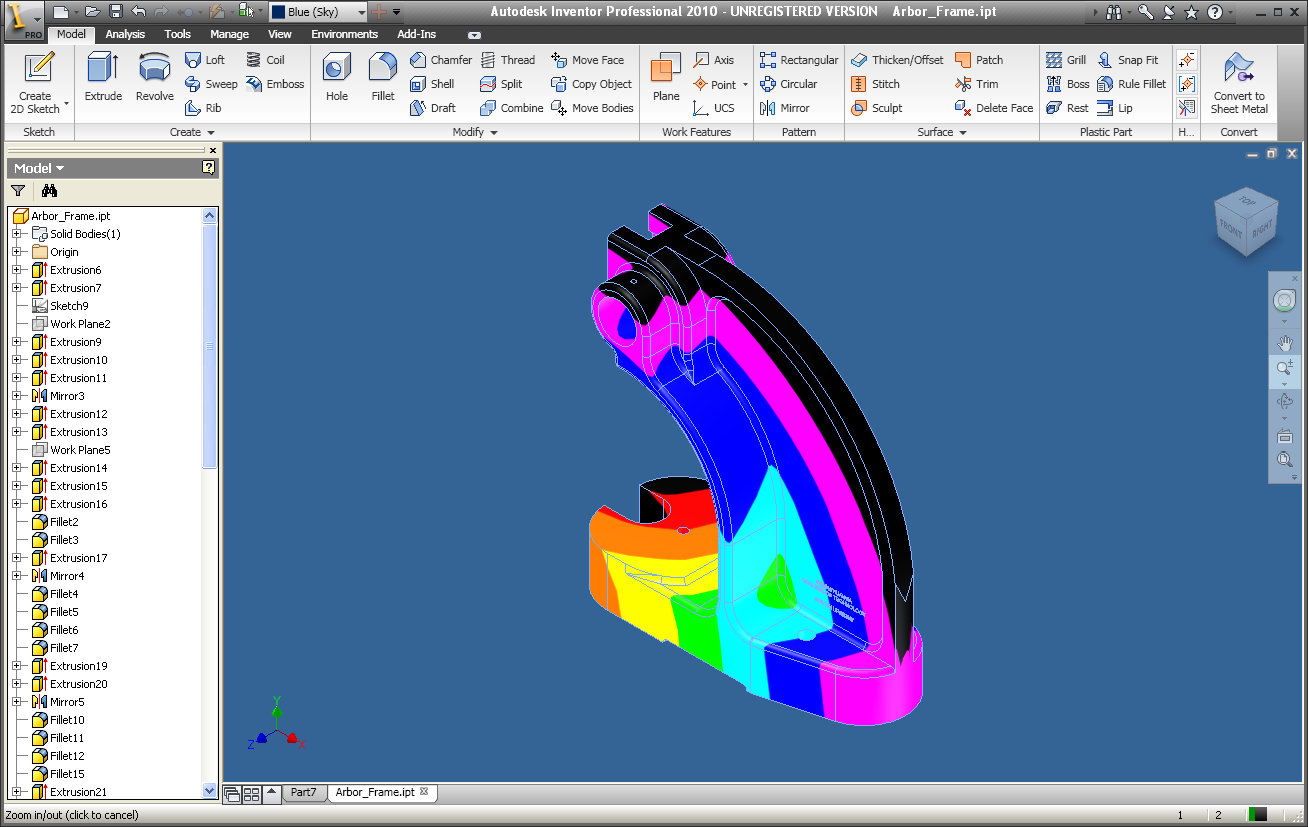


Fig 19 : Slicing

Fig 20 : Mapping

## Further References

**Inventor API help document**

* Objects, methods and properties
* Samples

## Sample code

The zip file provided with this document also contains some VBA and VB.Net examples.

* **Clientgraphics.ivb**

It contains VBA code, and can be loaded into the Inventor VBA environment via the VBA IDE. There are 3 modules and 1 class.

* modBasicSample

These macros demonstrate the creation of graphics primitives, using an index set, strip graphics and using native model geometry as a basis for generating client graphics. You can run the macros from the VBA IDE or from the Inventor user interface directly from the tools menu; Tools🡪Macros.

* modInteractionGraphics

These macros demonstrate client graphics of when associated with the InteractionEvents object.

* modAdvancedFun

This macro demonstrates some advanced client graphics functionality.

* clsDragComponent:

A utility class that is used with the InteractionGraphics sample.

Each macro is commented to help explain what it is doing, and how the task is accomplished.

* **VBNet\_ClientGraphics**

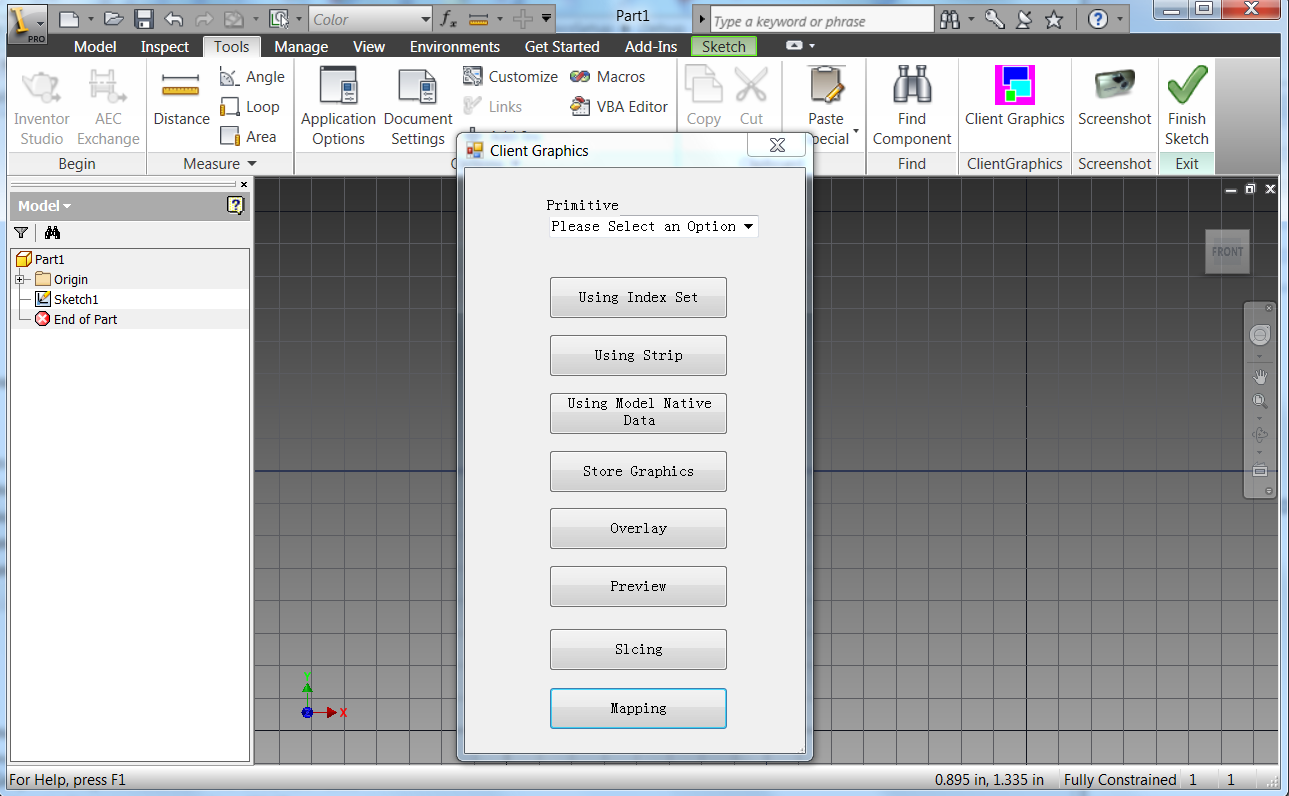
This is a VB.Net addin, which is accessed from Inventor from the Ribbon Bar.

* ClientGraphicsSetup.msi

This is the installer. After installing, open a Part, Assembly or Drawing document, then run the command from the user interface as follows (Select Tools🡪Client Graphics). This displays a dialog with a series of buttons that run each of the individual samples. They illustrate various techniques related to client graphics creation. Selecting an option in the CComBox will draw the relevant graphics primitive in the current view.

* Source code

The source code can be loaded and run in Microsoft’s Visual Studio development environment.



Thank you.

Developer Technical Services

Autodesk.