



Best Practices for Developing with ArcGlobe

Handling and Rendering Dynamic Content in 3D

Morakot Pilouk

Yaron Fine

ESRI Developer Network (EDN)

Outline



- Introduction
- Simple 3D Rendering on Globe
 - Out-of-the-box functionality
- Advanced 3D Rendering
 - APIs for Advanced 3D Rendering
 - GlobeGraphics API
 - GlobeDisplay API
 - Directly call OpenGL API
- Handling and Rendering Dynamic Content in 3D
 - Drawing from Tools and Commands
 - Drawing from Globe CustomLayer
- Dealing with Multi-threads
- Application Templates
- Demos
- Questions and Answers

Introduction



- Best Practices
 - Using applications in Dynamic GIS as our case study
 - Sharing our experiences
 - Pointing out programming pattern and things to avoid
- Dynamic GIS

Dynamic GIS



- **Dynamic Display**
 - of spatially related information
 - Visualizing changes in real-time in a continuous manner
 - Animated moving objects
 - Animated object representations
- **Dynamic Content**
 - Frequent update of the content
 - Location
 - Orientation
 - States
 - Attributes

Why Dynamic GIS?



- Technology driven
 - Real-time data capture
 - Data become more dynamic
 - Network bandwidth
 - Data streaming technology
 - Hardware accelerated display
 - CPU speed, system memory, etc
- Application requirements
 - Visualizing changes
 - Converging of GIS and other disciplines
 - Emergency response
 - Command and control
 - Real-time tracking
 - Simulation

Toward Supporting Dynamic GIS



- Enhance ArcGIS map rendering to support requirements for dynamic display environments
 - Fast display refresh
 - Continuous zoom/pan/roam
- Support moving significant numbers of display objects at sub-second refresh rates
 - Quickly draw objects to maintain visual continuity on
 - Change of location
 - Change of other characteristics
- Supporting dynamic query
 - Select/Identify moving objects
 - May involve spatio-temporal query (not part of today's scope)
- Supporting dynamic Analysis
 - Real-time overlay
 - Collision detection

Simple 3D Rendering on Globe



- Out-of-the-box functionality
- FeatureClass and FeatureLayer
- Animation Framework
- Globe Graphics 3D Toolbar
 - Point
 - Line
 - Text
 - 3D MarkerSymbols
- KML Layer

Globe 3D Graphics Toolbar and Symbol Picker



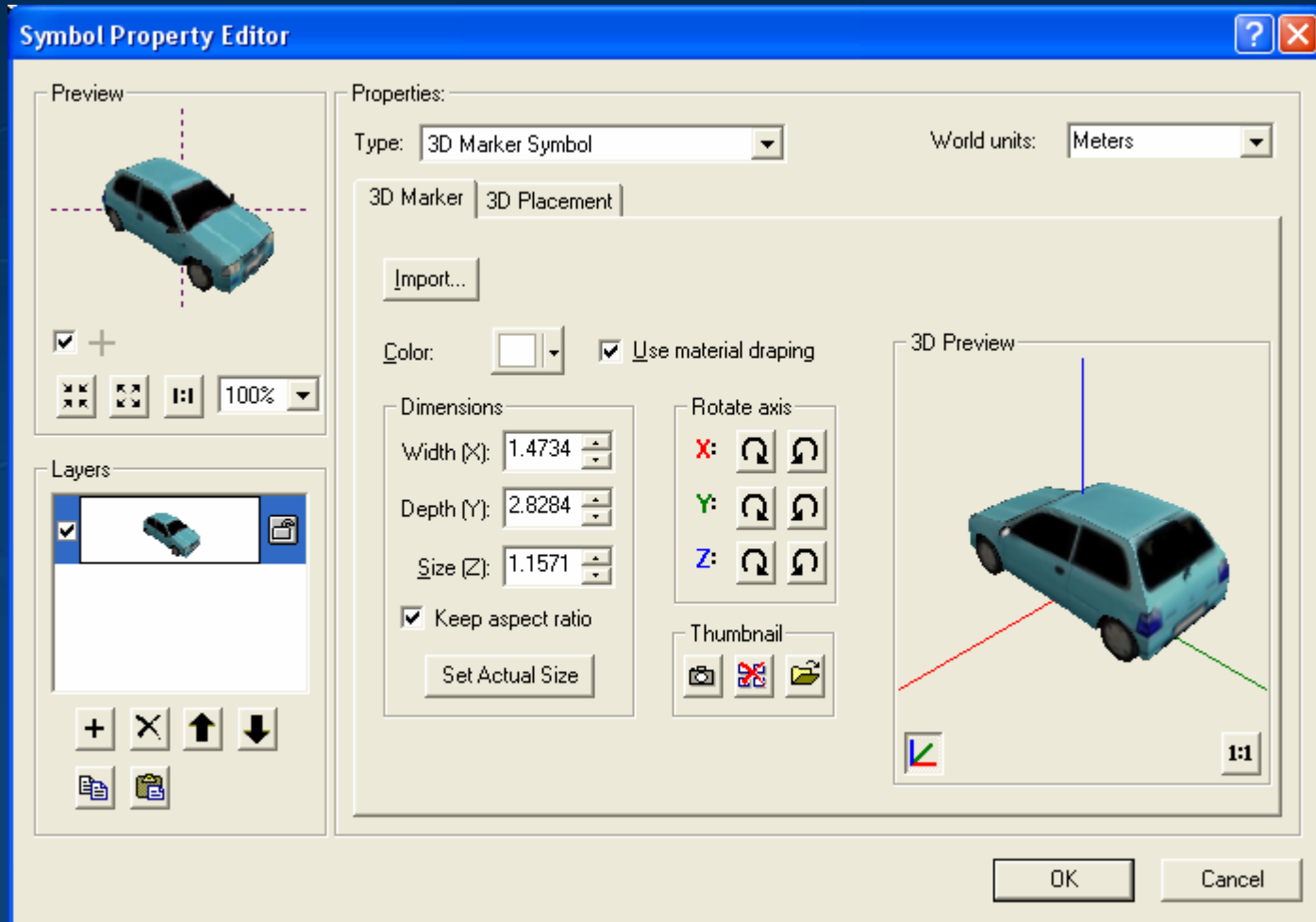
The screenshot displays the ArcGlobe application window titled "Untitled - ArcGlobe - ArcInfo". The interface includes a menu bar (File, Edit, View, Selection, Tools, Window, Help), a toolbar with various navigation and editing tools, and a layer list on the left showing "World Image".

Overlaid on the globe is the "Symbol Selector" dialog box. It features a "Category" dropdown set to "Urban Automobile". The main area contains a grid of 3D car symbols, each with a label: SUV4x4, Alto, AstonMartin, Audi, CadillacCatera, CadillacDeville, ChevyCavalier, ChryslerConcorde, DodgeIntrepid, Estate, FiatPanda, and GrandJeepCher... A "Preview" window on the right shows a 3D rendering of a teal car. Below the grid are "Options" for Color, Size (1.16), and Angle (0.00). Buttons for "Properties...", "More Symbols", "Save...", "Reset", "OK", and "Cancel" are at the bottom.

In the foreground, a "Default Symbol Properties" dialog box is open, showing three tabs: "Marker" (with a black dot symbol), "Line" (with a horizontal line symbol), and "Fill" (with a yellow square symbol). "OK" and "Cancel" buttons are at the bottom.

At the bottom of the ArcGlobe window, there are tabs for "Type", "Display", and "Source", and a status bar showing "Distance: 5618.531 Kilometers".

3D Symbol Property Editor





Advanced 3D Rendering

Advanced 3D Rendering



- Add functionality not available on the UI
 - Interactively displace/rotate graphic object
- Render unsupported data sources
 - Web Services, real-time feed, simulation
- Control the rendering as needed
 - Camera tracking moving object
- Add special effect to the globe display
 - Simulate fire, smoke, rain, flood, wave, etc



APIs for Advanced 3D Rendering

- **GlobeGraphics API**
 - GraphicsLayer
 - GraphicsElement
- **GlobeDisplay API**
 - 3D Symbols
 - Coordinate conversion utility
 - CustomGlobeLayer
- **OpenGL API**
 - Globe framework provides mechanism to plug-in OpenGL calls

GlobeGraphics API



- **GlobeGraphicsLayer**
 - Container of graphic elements
 - Similar to Map, Scene graphics layer
- **GraphicsElements**
 - Non-feature
 - Stay in memory at runtime
 - Serializable to disk with Globe document

GlobeGraphicsLayer



- Creating Layer and adding it to the Globe

```
//Create a new graphics layer
m_ipGlobeGraphicsLayer.CreateInstance(CLSID_GlobeGraphicsLayer);
ILayerPtr(m_ipGlobeGraphicsLayer)->put_Name(L"DynamicObjects");

//Add the new graphic layer to the globe
IGlobePtr ipGlobe;
m_ipGlobeDisplay->get_Globe(&ipGlobe);
IScenePtr(ipGlobe)->AddLayer(ILayerPtr(m_ipGlobeGraphicsLayer), VARIANT_TRUE);
```

- Activating the Layer

```
//Activate the new graphics layer
IScenePtr(ipGlobe)->ActiveGraphicsLayer(ILayerPtr(m_ipGlobeGraphicsLayer));
```

GlobeGraphicsElement



- Element types
 - Marker3DElement
 - LineElement
 - PolygonElement
 - MultiPatchElement
 - etc
- Creating Element
 - Setting Element Geometry and Properties
 - Symbolizing Element
 - Adding Element to the Graphics Layer (Container)
- Transforming and Updating Element

Creating GraphicsElement



```
//Create the element's geometry
IPointPtr ipPoint(CLSID_Point);
IZAwarePtr(ipPoint)->put_ZAware(VARIANT_TRUE);
ipPoint->PutCoords(position.longitude, position.latitude);
ipPoint->put_Z(position.altitude);

//Create the element's color (red)
IRgbColorPtr ipColor(CLSID_RgbColor);
ipColor->put_Red(255L);
ipColor->put_Green(0L);
ipColor->put_Blue(0L);

//Set the element's symbol
IMarkerSymbolPtr ipMarkerSymbol(CLSID_SimpleMarker3DSymbol);
ISimpleMarker3DSymbolPtr(ipMarkerSymbol)->put_Style(esriS3DMSSphere);
ISimpleMarker3DSymbolPtr(ipMarkerSymbol)->put_ResolutionQuality(1.0);
ipMarkerSymbol->put_Size(700.0);
ipMarkerSymbol->put_Color(IColorPtr(ipColor));

//Create the new marker symbol element
IElementPtr ipTrackElement(CLSID_MarkerElement);
IMarkerElementPtr(ipTrackElement)->put_Symbol(ipMarkerSymbol);
ipTrackElement->put_Geometry(IGeometryPtr(ipPoint));

//Add the graphic element to the graphics layer
IGraphicsContainerPtr(m_ipGlobeGraphicsLayer)->AddElement(ipTrackElement);
```



Updating GraphicsElement

- Updating geometry
- Updating properties

```
//Get the element by its index
IElementPtr elem;
IGraphicsContainer3DPtr(m_ipGlobeGraphicsLayer)->get_Element((long)m_trackObjectIndex, &elem);
//Get the geometry of the element
IGeometryPtr ipGeometry;
elem->get_Geometry(&ipGeometry);
//Update the element's position
IPointPtr(ipGeometry)->PutCoords((*position).longitude, (*position).latitude);
elem->put_Geometry(ipGeometry);

//Set the element's properties
IGlobeGraphicsElementPropertiesPtr ipProps(CLSID_GlobeGraphicsElementProperties);
ipProps->put_UseCallOutLine(VARIANT_FALSE);
ipProps->put_OrientationMode(esriGlobeGraphicsOrientationLocal);
ipProps->put_Illuminate(VARIANT_TRUE);
ipProps->put_FixedScreenSize(VARIANT_TRUE);
ipProps->put_DrapeElement(VARIANT_FALSE);

//Update the element in the graphic layer
m_ipGlobeGraphicsLayer->SetGlobeProperties(elem, ipProps);
m_ipGlobeGraphicsLayer->UpdateElementByIndex((long)m_trackObjectIndex);
```



Applying Transformation

- Translation in X, Y, Z
- Scaling in X, Y, Z
- Rotation around X, Y, Z

//C# Sample

```
//Get the element from the graphics container
```

```
IGlobeGraphicsLayer globeGraphicsLayer = m_graphicsCont3D as IGlobeGraphicsLayer;  
int index;  
globeGraphicsLayer.FindElementIndex(m_movingElement, out index);
```

```
//Get the current element transformations
```

```
IVector3D translation, scaling, rotation;  
globeGraphicsLayer.GetElementTransformation(index, out translation, out scaling, out  
rotation);
```

```
//Set the new transformations
```

```
translation.SetComponents(tx, ty, tz);  
scaling.SetComponents(sx, sy, sz);  
rotation.SetComponents(rx, ry, rz);  
globeGraphicsLayer.SetElementTransformation(index, translation, scaling, rotation);
```

Pros and Cons



- Ease of use – high level programming
- Index VS Reference
- Performance implication

Demo



- Globe Graphics
- Tracking Dynamic Object

Using GlobeDisplay API



- Directly draw geometry using selected symbol through IDisplay, IDraw, IGlobeDisplay3
- Allows direct call to OpenGL API
- When to use
 - In Commands and Tools
 - Inside the BeforeDraw() and AfterDraw()
- How to use
 - Set the Display into Direct OpenGL mode
 - Set the symbol to the Display
 - Call the method to draw the geometry
 - Optionally call the OpenGL functions
 - Restore the Display mode
- Pros and Cons

Directly call OpenGL API



- Framework for OpenGL plug-in
 - In the method BeforeDraw() and AfterDraw() of IGlobeDisplayEvents
 - Methods of IDisplay and IDraw can be mixed with OpenGL calls when the DirectOpenGLDraw flag of the IGlobeDisplay3 is set to True
 - In the method DrawImmediate() of the ICustomGlobeLayer



Handling and Rendering Dynamic Content in 3D

Drawing from Tools and Commands



- Attach drawing to mouse, keyboard, joystick events
- Good for custom drawing that doesn't require complex object management
- Should listen to `IGlobeDisplayEvents`
- OpenGL calls should be within the `BeforeDraw()` or `AfterDraw()`
 - OpenGL transformation is only valid in these two methods

Demo



- Digitizing on the globe

Drawing from Custom Layer



- Good for rendering and handling considerable amount of data from:
 - Unsupported data sources, e.g.
 - Remote Servers
 - Web Services
 - Dynamic feed with frequent update rate:
 - Tracking Services, e.g. Flight Direct
 - Data streaming via network ports
 - Simulation data, e.g. MAK VRLink, Satellite orbits

Implementing custom layer



- Method 1: Listen to `GlobeDisplayEvents` and do all the OpenGL calls in the `BeforeDraw()` and `AfterDraw()` method
- Method 2: Implementing `ICustomGlobeLayer` interface
- Implement all other mandatory interfaces to make a valid Globe layer:
 - `ILayer`, `ILayerInfo`, `ILayerExtensions`, `ILegendInfo`
 - `IGeoDataset`
 - `IPersistStream`, `IPersistVariant`
 - etc

Connecting to GlobeDisplayEvents



- Start listening to GlobeDisplayEvents
 - In Custom globe layer avoid calling this in the constructor or FinalConstruct()

```
HRESULT CMyCommand::StartListeningToGlobeDisplayEvents(IGlobeDisplay* pGD)
{
    IUnknownPtr ipUnk(this);
    return AtlAdvise(pGD, ipUnk, __uuidof(IGlobeDisplayEvents), &m_dwCookie);
}
```

- Stop listening to GlobeDisplayEvents
 - In Custom globe layer avoid calling this in the FinalRelease() or destructor

```
HRESULT CMyCommand::StopListeningToGlobeDisplayEvents(IGlobeDisplay* pGD)
{
    return AtlUnadvise(pGD, __uuidof(IGlobeDisplayEvents), m_dwCookie);
}
```



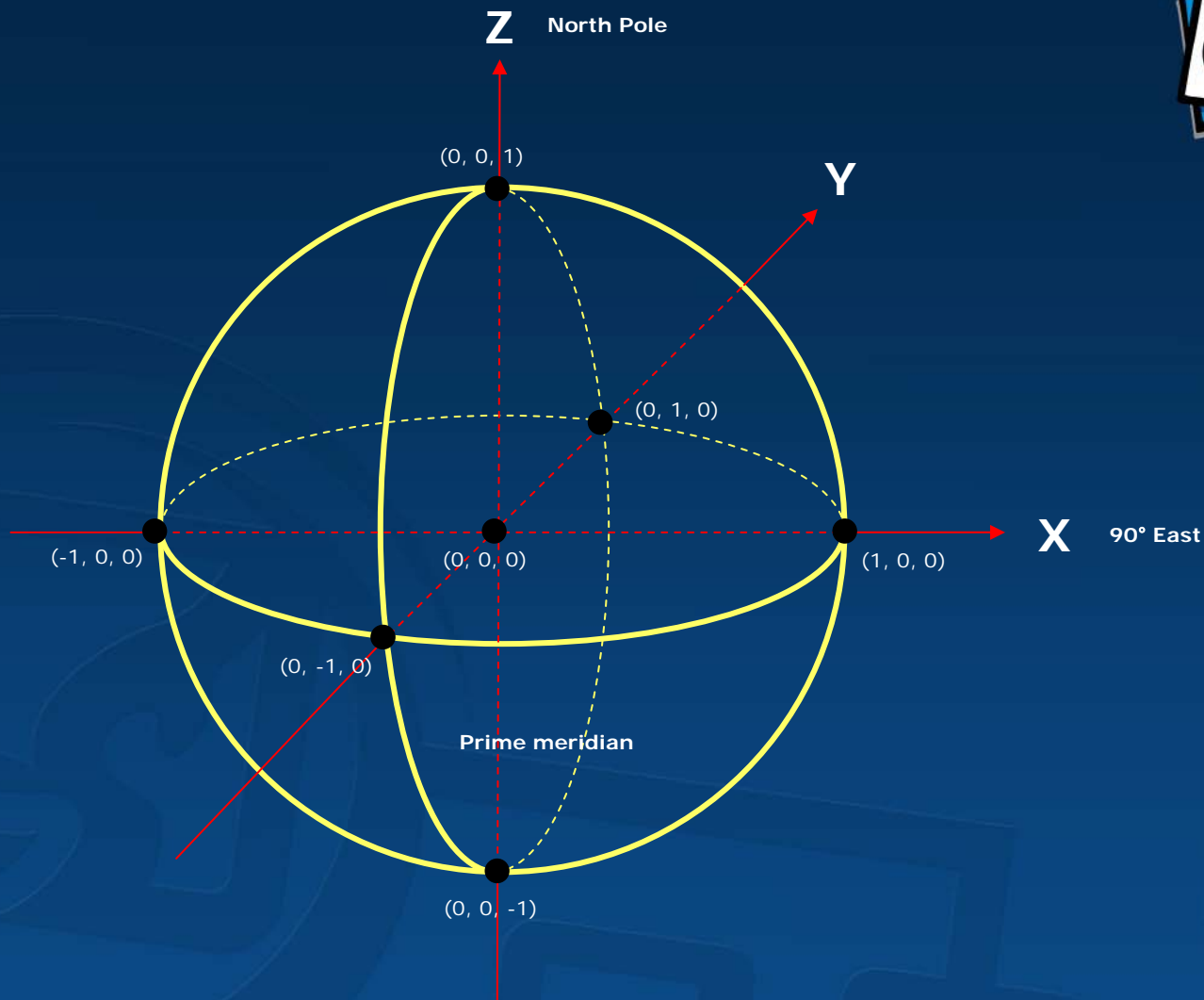
ICustomGlobeLayer Interface

- Available with 9.2 Release
- No need to listen to GlobeDisplayEvents
- OpenGL can be directly called inside the method DrawImmediate()
- DrawType should be set to esriGlobeCustomDrawOpenGL
- Provides Hit-Test mechanism for picking and selection in 3D
- Provides mechanism to get around floating point rendering precision in OpenGL

Drawing 3D Objects on Globe



- Globe Coordinate Systems
 - Geographic (WGS 84)
 - Geocentric (internal globe)
 - Coincides with OpenGL transformation
 - Window/Viewport (2D)



Normalized spherical-based geocentric coordinate system

Globe Coordinate Transformation



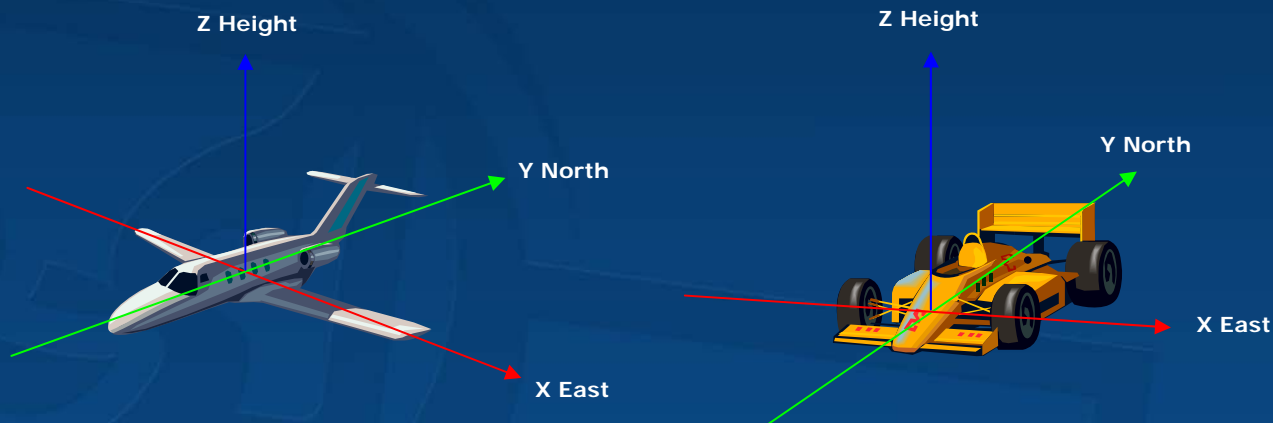
- New utility interface (IGlobeViewUtil) in 9.2 Release for coordinate transformation

```
//Declaring the GlobeViewUtil
IGlobeViewUtilPtr  m_ipGlobeViewUtil;
//QI the GlobeViewUtil from the camera
ICameraPtr ipCamera;
hr = m_ipSceneViewer->get_Camera(&ipCamera);
m_ipGlobeViewUtil = ipCamera;
//Convert a geographic coordinate to geocentric
m_ipGlobeViewUtil->GeographicToGeocentric(longitude, latitude, altitudeMeters, &X, &Y, &Z);
//Convert geocentric to geographic
m_ipGlobeViewUtil->GeocentricToGeographic(X, Y, Z,
&longitude, &latitude, &altitudeMeters);
//Convert geocentric to window
int winX, winY;
m_ipGlobeViewUtil->GeocentricToWindow(X, Y, Z, &winX, &winY);
```

Object's Local Coordinate System



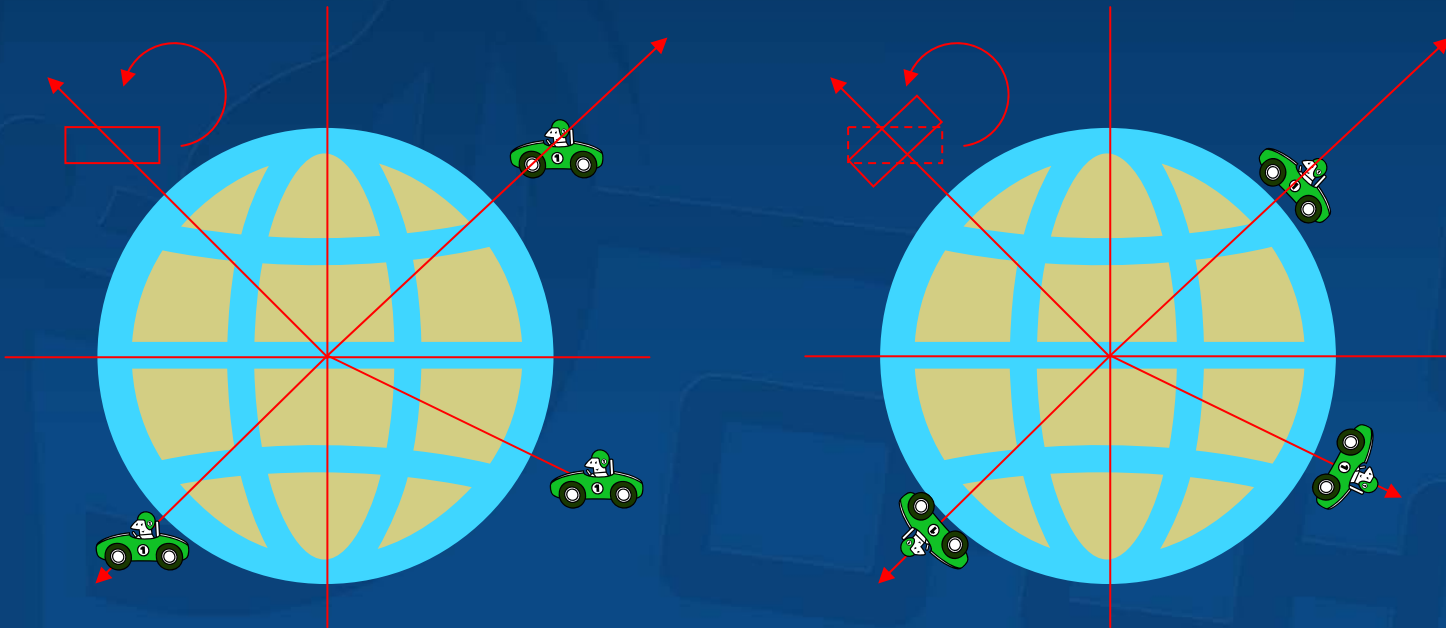
- Object's coordinate should be properly aligned with its local coordinate system



Transforming from Local to Globe



- Typically, the Z-axis of the object should be aligned with the globe radius vector



Using 3D Model From File



- 3D Model in OpenFlight (.FLT) or 3D Studio (.3DS) can be imported into Marker3DSymbol
- IMarker3DSymbol provides a method CreateFromFile() for this purpose
- Two approaches can be used:
 - Set the symbol to MarkerElement and add the MarkerElement to the GlobeGraphicsLayer
 - Create OpenGL Display List from the Marker3DSymbol to be reused

Creating a DisplayList from Marker3DSymbol



```
IGlobeDisplay3Ptr(ipGlobeDisplay)->put_DirectOpenGLDraw(VARIANT_TRUE);
glMatrixMode(GL_MODELVIEW);
IGlobeDisplayRenderingPtr ipGlobeDisplayRend(ipGlobeDisplay);
double globeRadiusMeters;
ipGlobeDisplayRend->get_GlobeRadius(&globeRadiusMeters);
double scale = 1.0/globeRadiusMeters; // normalized by the Globe Radius
int intSymbolDisplayList = glGenLists(1);
glNewList(intSymbolDisplayList, GL_COMPILE);
{
    glDisable(GL_COLOR_MATERIAL);
    glPushMatrix();
    {
        glScaled(scale, scale, scale);
        IDisplayPtr ipDisplay(ipGlobeDisplay);
        if (ipDisplay != NULL)
            ipDisplay->SetSymbol(ISymbolPtr(ipMarker3DSymbol));
        else
            return;
    }
    glPopMatrix();
    glEnable(GL_COLOR_MATERIAL);
}
glEndList();
IGlobeDisplay3Ptr(ipGlobeDisplay)->put_DirectOpenGLDraw(VARIANT_FALSE);
```

Transforming from Local to Globe



```
HRESULT CalculateSymbolOrientation(ISceneViewer* pSceneViewer, double glX, double
glY, double glZ, WKSPointZ *orientation)
{
    m_ipVector3D->SetComponents(glX, glY, glZ);
    double inclination, azimuth;
    m_ipVector3D->get_Inclination(&inclination);
    m_ipVector3D->get_Azimuth(&azimuth);
    (*orientation).X = (-1.0 * RAD2DEG(inclination)) + 180.0;
    (*orientation).Y = 0.0;
    (*orientation).Z = 180.0 - RAD2DEG(azimuth);
    return S_OK;
}

void TranslateAndRotate(double glX, double glY, double glZ, WKSPointZ
symbolOrientation, double udfAzimuth)
{
    glTranslated(glX, glY, glZ);
    glRotated(symbolOrientation.Z, 0.0, 0.0, 1.0);
    glRotated(symbolOrientation.X, 1.0, 0.0, 0.0);
    glRotated(symbolOrientation.Y, 0.0, 1.0, 0.0);
    glRotated(udfAzimuth, 0.0, 1.0, 0.0);
}
```



The Drawing Sequence

- Call `glPushMatrix` to save the matrix stack
- Reproject object's coordinate into WGS1984 (if needed)
- Convert from WGS 84 to geocentric coordinate system
- Align the object Z-axis to the globe radius vector
- Translate the object into its place and orient it as needed
- Scale the object so that it will be rendered at an appropriate size
- Call the display list of the relevant symbol in order to draw it
- Call `glPopMatrix` in to restore the matrix stack

Simple Drawing Sequence



```
STDMETHODIMP CMyLayer::DrawImmediate(IGlobeViewer * pGlobeViewer)
{
    //Make sure that the layer is visible and valid
    if(VARIANT_FALSE == m_bVisible || VARIANT_FALSE == m_bValid)
        return S_OK;
    ...
    double glX, glY, glZ;
    WKSPointZ wksSymbolOrientation;
    . . .
    //Translate the object into place
    glPushMatrix();
    {
        //Get the OpenGL coordinate of the object
        m_ipGlobeViewUtil->GeographicToGeocentric(obj->dLongitude,
                                                    obj->dLatitude,
                                                    obj->dAltitudeMeters,
                                                    &glX, &glY, &glZ);

        //Calculate the object's orientation
        CalculateSymbolOrientation(ipSceneViewer, glX, glY, glZ,
                                   &wksSymbolOrientation);

        //Translate the object into its place and orient it
        TranslateAndRotate(glX, glY, glZ, wksSymbolOrientation, dblSymbolAngle);
        //Set the object's scale
        glScaled(dScale,dScale,dScale);
        //Draw the object
        glCallList(m_intMainSymbolDisplayList);
    }
    glPopMatrix();
    return S_OK;
}
```

Optimizing the Drawing



- Minimize the computation load inside the DrawImmediate()
- Apply different level of details
 - Far away object should use a simplified symbol
- Filter out objects which are outside the display viewport
- Use a timer for the display redraw
 - Apply a common timer in case of multiple layers
 - In .NET, timer event handler occurs on a different thread. Lookout of cross-apartment call causing deadlock or poor performance
 - Use Invoke() to get around this problem

Managing Dynamic Objects



- Considerations
 - Rendering performance
 - Data streaming rate
 - Data update rate
 - Spatial Distribution
 - Threading model to avoid rendering delay
 - Data update thread
 - Rendering thread
 - Layer functionality
 - Querying
 - Selection
 - Identify
 - Editing
 - Analysis
 - Recording and Playback
 - Etc
 - Development environment
 - C++ for CustomLayer
 - .NET for UI components

Connecting to Real-time Feed



- Available data feeds
 - ArcGIS Tracking Services
 - Flight Direct
 - Peer to peer communication (UDP, TCP)
 - MAK VRLink Simulation
 - Satellite Orbits Simulation
 - WSDL (SOAP)
 - ArcWeb
 - NOAA weather service
 - RSS-Feed
 - Yahoo
 - etc
- Mode of connection
 - Push mode
 - Pull mode

Labeling Object in Globe

- Choosing fonts for performance
 - Outline font
 - Is vector
 - Can have depth
 - Can be rotated freely
 - More expensive to render
 - Bitmap font
 - Is raster
 - No depth
 - Cannot be rotated
 - Cheaper to render
 - Texture font
 - Similar to Bitmap font, but can be rotated
- Text Billboarding
 - Always faces camera (viewer) for legibility

Implementing Hit Test



- For supporting picking and selection in 3D
- Need to implement code in the `ICustomGlobeLayer::Hit()`
- Assign the unique name to the object by calling `glLoadName()` before calling OpenGL to draw an object
 - Globe will pass this name back as the `hitObjectID` parameter of the `Hit()` method above
 - This unique name can be used to retrieve the object's instance from the layer

Keeping Dynamic Object in Motion



- Cache enough information to facilitate computation of object location
 - Last location
 - Speed
 - Heading
 - Orientation
 - Acceleration
 - Drawing interval
 - etc
- Estimate the new location of the object before each drawing cycle

Dealing with Multi-threads



- Thread and Apartment
 - COM threading model
 - Marshalling/Serialization
 - Avoiding cross apartments call
- Globe uses two threads
 - Cache update thread
 - Rendering thread
- Globe layer will be cloned multiple times
 - Multiple instances during runtime
- Implementing IPersistStream or IPersistVariant to allow cloning
 - Pay attention to what to be persisted/serialized
 - Implementing Save() and Load()

Application Templates



- Bringing best practices into templates
- Making implementation easier
 - Create start up code automatically
- Available templates
 - .NET BaseClass custom globe layer
 - Developers only need to override DrawImmediate()
 - .NET IDE integration
 - Custom globe layer
 - Globe Control Application
 - Commands, Tools, Toolbars

Demos



- Weather Layers
- Flight Direct Tracking Service
- Satellite Simulation
- MAK VRLink
- .NET IDE Integration

What Next?



- Improving Globe Core API
 - Annotation in 3D
 - Dynamic 3D symbols
- Globe developer contents on EDN
 - Technical articles
 - Samples
 - Snapshots
 - Videos

Additional Resources



- <http://edn.esri.com>
 - 3D Analyst Technical Document
- Online OpenGL Resources
 - OpenGL.org
 - GameDev/NeHe
- Books
 - OpenGL Super Bible

Session Evaluations Reminder



Session Attendees:
Please turn in your session evaluations.

... Thank you