RenderCache API

The RenderCache API allows SDK developers to access the same rendering data as the internal MODO renderer.
The render cache exposes the geometry cache and shading information.

The renderer is not accessing items directly from the scene, it's using the Tableau interface to access the "baked" data. The renderer samples the data from the Tableau, however, during the data sampling, the renderer will create its own data which is not stored in the Tableau, such as fur & hair data. Therefore we've added a RenderCache API which allows SDK developers access the whole render data.

Geometry cache has data is grouped into ILxGeoCacheSurface(s) which have data stored into ILxGeoCacheSegment(s).

For example, if we have a scene and it contains one mesh item which has multiple materials applied to it, the geo cache will have one ILxGeoCacheSurface for each material.
If you need to know which item in the scene was used to "generate" the ILxGeoCacheSurface, you can use the ILxGeoCacheSurface::SourceItem() method
and helper method CLxUser_GeoCacheSurface::GetSourceItem().
In the above case, that method will return the same item for all geo cache surfaces.

Lets look at the case where there are instanced items in the scene.
In this case only one geo cache surface is going to have geometry segments stored and all other geo cache surfaces will be "instanced". In other words, they will have only instance information (transform, shaders) stored and a reference to the geo surface that holds the geometry information.
To check if surface is instanced, use the ILxGeoCacheSurface::IsInstanced(), and use ILxGeoCacheSurface::SourceSurface() to obtain the geometry data from instanced surface (or helper method CLxUser_GeoCacheSurface::GetSourceSurface()).
Every instanced surface will have an index; use ILxGeoCacheSurface::InstanceIndex() to query for it.
NOTE: For proxy items there can be multiple ILxGeoCacheSurfaces created based on if proxy item has different transform than proxied scene.
For example if we create a cube mesh and create a proxy item for that cube. The render cache will report only one surface in that case. However, if translate the item, the render cache will report 2 surfaces.

The renderer doesn't make a distinction between replicated or instanced items, they're all "seen" as instances by the renderer, i.e. they're stored as ILxGeoCacheSurface(s) in the render cache.

The geometry data is stored in the geo cache segments. When geo cache is created, the data is not yet stored, i.e. the geo segments won't be created. The segments are created/populated by calling ILxGeoCacheSurface::LoadSegments(). The segments are unloaded with ILxGeoCacheSurface::UnloadSegments(). This gives developer control on what data needs to be stored in the memory. Make sure that when accessing the geo surface geometry, i.e. when LoadSegments is invoked, that render cache is not in update mode.

The segments can store triangles and line data only.
The vertex data is grouped in "vertex features" (world space position, world space normal etc).
The vertex data can be extracted either by polygon-per-vertex (like a triangle soup) or by vertex and polygon vertex indices. In later case the segment will store unique vertices and generate the polygon vertex indices into unique vertex data. The unique vertex list is only generated for surfaces that hold the triangle data.
Have a look at ILxGeoCacheSegment::GetPolygonVertexIndices(). Note that this can be an expensive operation on a segments that have a lot of geometry.

Each instance of ILxRenderCache, ILxGeoCacheSurface and ILxGeoCacheSegment is reference counted, and once there are no references to it, it will be released. To clear all the render cache data use the ILxRenderCache::Clear().

RenderCache can by dynamically updated. To monitor the changes in the render cache you can use the ILxRenderCacheListener. Also, because of reference counting and dynamic updates make sure you're not manually changing the reference counts or holding a reference to geo cache surface (or geo cache segments), otherwise the dangling reference could potentially lead to unexpected behavior.

ILxRenderCacheService
This service is used to create the render cache.

```
LxResult CreateRenderCache (
      LXtObjectID         self,
      void              **ppvObj,
      unsigned int        createFlags);
```

Use this method to create a render cache.
There's also a user class helper method (in CLxUser_RenderCacheService)

```
      bool
NewRenderCache  (
      CLxLoc_RenderCache    &rcache,
      unsigned int          createFlags);
```

Argument "createFlags" specifies the behaviour of render cache.
On render cache creation all the geometry surfaces are created, but their data (vertex position, etc.) will be populated on request. This gives caller control on how much memory is in use.

You can specify if render cache should generate fur data for geo cache surfaces and if geo cache surface data should contain displaced data.

```
      LXfRENDERCACHE_GEOCACHE_DISPLACE
      LXfRENDERCACHE_GEOCACHE_GENFUR
```

Use `LXfRENDERCACHE_ FULL` flag if you want both displaced and furry surfaces.

The caller can specify if render cache should track current scene (scene selection events) by using the flag `LXfRENDERCACHE_TRACK_CURRENT_SCENE` .
***NOTE**:*
Currently the render cache is always tracking the current scene and that can't be turned off.

Render cache updates automatically by default, i.e. whenever there's a scene change the render cache will be automatically updated.
Caller can set `LXfRENDERCACHE_TURN_OFF_AUTO_UPDATES` to turn off this behavior.

Sometimes the render cache is not picking up the updates from certain scene items (various procedural items). To force full render cache rebuild caller can use flag `LXfRENDERCACHE_FORCE_FULL_UPDATE` .

ILxRenderCache

By default render cache is tracking the changes in the scene including the current scene time and it automatically updates itself.

```
      void
Clear (
      LXtObjectID             self);
```

Use Clear() method to clear the render cache.

```
      LxResult
GeoSurfaceCount(
      LXtObjectID              self,
      int                     *count);
```

Returns number of geo cache surfaces.

```
      LxResult
GeoSurfaceAt (
      LXtObjectID               self,
      int                       index,
      void                    **srf);
```

Returns the geo cache surface at given index.

There's also a user class method:

```
      bool
GetGeoSurface  (
      int                     index,
CLxLoc_GeoCacheSurface        &srf)
```

ILxRenderCacheListener
The listener notifies caller that render cache has been changed.

```
      void
RenderCacheDestroy (
      LXtObjectID           self);
```

Called when render cache is being destroyed. This happens when user changes the current scene (if `LXfRENDERCACHE_TRACK_CURRENT_SCENE` was specified).

```
      void
UpdateBegin (
      LXtObjectID           self);

      void
UpdateEnd (
      LXtObjectID           self);
```

UpdateBegin() is called prior modification of render cache, UpdateEnd() is called after render cache has been updated. This is used to notify render cache clients that render cache is in volatile state. For example you shouldn't call GeoCacheSurface::LoadSegments() during render cache update.

```
      void
GeoCacheSurfaceAdd (
      LXtObjectID           self,
      LXtObjectID           geoSrf);
```

Called after the geo surface has been added to cache.

```
      void
GeoCacheSurfaceRemove (
      LXtObjectID           self,
      LXtObjectID           geoSrf);
```

Called before the geo surface has been removed from cache.

```
      void
GeoCacheSurfaceGeoUpdate (
      LXtObjectID           self,
      LXtObjectID           geoSrf);
```

Called when the geo surface data has been changed (this method is called only for non-instanced surfaces).

```
      void
GeoCacheSurfaceXformUpdate  (
      LXtObjectID           self,
      LXtObjectID           geoSrf);
```

Called when the xform data has been changed.

```
      void
GeoCacheSurfaceShaderUpdate  (
      LXtObjectID           self,
      LXtObjectID           geoSrf);
```

Called when the shading data has been changed.

ILxGeoCacheSurface

The geometry data is organized in surfaces which contain segments. All segments in one surface have same bounding box, polygon type (triangles, or lines), shader, transform (for shutter open and close).

```
      LxResult
ShaderMaskName  (
      LXtObjectID         self,
      const char        **name);
```

Returns the shader mask name of the surface. Please note that name is not unique, use ID() to get unique ID for a surface.

```
      int
ShaderMaskType(
      LXtObjectID         self);
```

Returns one of LXi_SURF_XXX flags.

```
      LxResult
SourceItem  (
      LXtObjectID         self,
      void              **ppvObj);
```

Return source item for this surface. Iinstanced and/or replicated will have a source item.
There's also a user method:

```
      bool
GetSourceItem  (
      CLxLoc_Item       &item);

      int
IsInstanced  (
      LXtObjectID         self);
```

Returns 1 if surface is instanced.

```
      int
InstanceIndex  (
      LXtObjectID               self);
```

Returns surface instance index, if surface is not instanced the returned value is -1.

```
      LxResult
SourceSurface (
      LXtObjectID             self,
      void                   **ppvObj);
```

This mehtod returns the source geo cache surface which contains the segment data.
There's also a user class method:

```
      bool
GetSourceSurface  (
      CLxLoc_GeoCacheSurface      &srf)
```

Note that render cache doesn't make a difference between instanced or proxied surfaces (i.e. render cache only works with surfaces and their instances).

```
      LxResult
GetBBox (
      LXtObjectID            self,
      LXtBBox               *bbox);
```

Returns the surface bounding box size (for all segments).

```
      LxResult
GetXfrm (
      LXtObjectID      self,
      LXtVector        pos,
      LXtMatrix        rot,
      LXtVector        scl,
      int              endpoint);
```

This method returns transform information, endpoint can be used to specify at which point in time you want to get transform for, shutter open = 0, shutter close = 1.

```
      void
SegmentCount  (
      LXtObjectID            self,
      int                   *count);
```

Returns the number of segments in the surface.
Note this method will return 0 if segments are not loaded.

```
      void
PolygonCount  (
      LXtObjectID            self,
      int                   *count);
```

Returns the total number of polygons in the surface.
Note this method will return 0 if segments are not loaded.

```
      void
VertexCount  (
      LXtObjectID            self,
      int                   *count);
```

Returns the total number of vertices in the surface.
Note this method will return 0 if segments are not loaded.

```
      LxResult
SegmentAt  (
      LXtObjectID        self,
      int                index,
      void              **segment);
```

Returns the geo cache segment at given index.
Note this method will return LXe_NOTFOUND if segments are not loaded.
There's also a user method available:

```
      bool
GetSegment  (
      int                          index,
      CLxLoc_GeoCacheSegment       &seg);

      LxResult
VisibilityFlags  (
      LXtObjectID                   self,
      LXpGeoCacheSrfVisibility     *flags);
```

Return visibility flags for this surface.
LXpGeoCacheSrfVisibility contains the flags which specify if surface is visible for example in camera rays, etc.

```
struct LXpGeoCacheSrfVisibility
{
      unsigned   camera     : 1;
      unsigned   indirect   : 1;
      unsigned   reflection : 1;
      unsigned   refraction : 1;
      unsigned   subscatter : 1;
      unsigned   occlusion  : 1;
};

      unsigned,
ID (
      LXtObjectID             self);
```

Each surface has a unique id which can be obtained by calling ID() method.

```
      LxResult
LoadSegments (
      LXtObjectID             self);
```

If render cache was initialized with populate on demand flag then caller can use LoadSegments() method to load all the segments for the surface. If this method is invoked during render cache update (look at the ILxRenderCacheListener::UpdateBegin() and UpdateEnd()) the method will return LXe_NOTREADY.

```
      LxResult
UnloadSegments) (
      LXtObjectID             self);
```

Unload segment data. If this method is invoked during render cache update (look at the ILxRenderCacheListener::UpdateBegin() and UpdateEnd()) the method will return LXe_NOTREADY.

```
      ILxTableauVertexID
GetVertexDesc (
      LXtObjectID             self);
```

Return's tableau vertex which can be used to obtain vertex feature names. For example, if you need vertex feature name for a UV, if there are multiple UVs present in the segment data.

These methods will return material, part or pick polygon tag name.

```
      const char*,
MaterialPTag   (
      LXtObjectID             self);

      const char*
PartPTag   (
      LXtObjectID             self);

      const char*
PickPTag  (
      LXtObjectID             self);
```

ILxGeoCacheSegment

```
      LxResult
GetBBox  (
      LXtObjectID             self,
      LXtBBox                *bbox);
```

Return the segment's bounding box in the world-space.

```
      void
PolygonCount  (
      LXtObjectID             self,
      int                    *count);
```

Return polygon count for this segment.

```
      void
VertexCount)  (
      LXtObjectID             self,
      int                    *count);
```

Return number of vertices for this segment.

```
      void
VertsPerPoly  (
      LXtObjectID             self,
      int                    *count);
```

All polygons in the segments have same number of vertices in the polygon.
It can be 3 for triangles, 2 for curves and (hair & fur).

```
      void
VertexFeatureCount  (
      LXtObjectID             self,
      int                     feature,
      int                    *count);
```

Returns if the vertex feature is stored in the segment.
OPOS and ONRM are always stored.
The count for WVEL, RAD and FUR will be either 0 or 1.
For UV, DPDU and DPDV the count will return how many UV vertex maps are stored in the segment.

There are two ways to access the vertex data from segment. Per-polygon-per-vertex, or per-vertex.

The feature should be initialized to one of these:

```
LXiRENDERCACHE_GEOVERT_OPOS  Object-space position (LXtFVector)
LXiRENDERCACHE_GEOVERT_ONRM  Object-space normal (LXtFVector)
LXiRENDERCACHE_GEOVERT_OVEL  Object-space velocity (LXtFVector)
LXiRENDERCACHE_GEOVERT_RAD Vertex radius (float)
LXiRENDERCACHE_GEOVERT_FUR  Fur params(LXtFVector [U,V(=lenParm), id])

LXiRENDERCACHE_GEOVERT_UV  Vertex UV coords. (LXtFVector2)
LXiRENDERCACHE_GEOVERT_DPDU  Vertex UV derivatives (LXtFVector3)
LXiRENDERCACHE_GEOVERT_DPDV  Vertex UV derivatives (LXtFVector3)
```

The fur data is stored either as poly lines (cylinders) or triangles, depending on the ‘Strip’ option on Fur Material. The fur parameters are U,V,ID.
The fur fibers are stored contiguously grouped by same ID and in order from V=0.0 to 1.0.
FUR and RAD vertex features are only available when cylinders primitive is used, they’re not available for triangle furry surfaces.

```
      LxResult
GetPolygonVertexFeature  (
      LXtObjectID       self,
      int               feature,
      void             *featureData,
      int               count,
      int               start);
```

Segment vertex feature data is copied to array provided by caller in a per-polygon-per-vertex fahsion (like a triangle soup). Caller can specify the start offset into the segment's data.
This can be useful if caller wants to keep the same buffer size to get the segment data.
The start and count are in number of elements which are from 0 to VertsPerPoly() * PolygonCount() (not in byte size).

```
      LxResult
GetVertexFeature  (
      LXtObjectID       self,
      int               feature,
      void             *featureData,
      int               count,
      int               start);
```

Segment vertex feature is copied per-vertex into client's memory. Caller can specify the start offset into the segment's data. The start and count are in number of elements which are from 0 to VertexCount() (not in byte size).

```
      LxResult
GetPolygonVertexInds (
      LXtObjectID            self,
      int                   *polyVertexInds,
      int                    count,
      int                    start);
```

Copy polygon vertex indices into client's memory. This is usefull if client want's to get only the vertex data and have indices into the vertex data arrays.
Caller can specify the start offset into the segment's data. The start and count are in number of elements which are from 0 to VertsPerPoly() * PolygonVertexCount() (not in byte size).
NOTE: `GetPolygonVertexInds()` only works with triangle data.

There's a SDK sample called rendercache which shows in detail how the RenderCache API can be used.

Changes:

MODO 10.2

- Fixed instanced surfaces update problems

May 2015

- Removed MaterialItem from ILxGeoCacheSurface

23rd March 2015

- Added UpdateBegin() and UpdateEnd() to **ILxRenderCacheListener**
- Added Time() and Update() to **ILxRenderCache**
- Modified ILxGeoCacheSurface::LoadSegments() and UnloadSegments()

23rd Jan 2015

- **ILxRenderCacheService**
  - changed CreateRenderCache() to include create flags

- **ILxRenderCache**
  - removed Init()
  - removed SetTime()
  - removed Update()
  - removed GeoEnum()
  - added GeoSurfaceCount()
  - added GeoSurfaceAt()

- **ILxRenderCacheGeoVisitor** has been removed.

- Added **ILxRenderCacheListener**

- **ILxGeoCacheSurface**
  - renamed Name() to ShaderMaskName()
  - Added UnloadSegments()

- **ILxGeoCacheSegment**
  - removed Load()
  - removed Unload()
  - renamed SegmentByIndex() to SegmentAt()
  - `GetPolygonVertexInds()` only works with triangle data now

Known Issues

- RenderCache API is not multi-threaded safe
- CLxLoc_GeoCacheSurface::MaterialPtag(), CLxLoc_GeoCacheSurface::PartPtag() and CLxLoc_GeoCacheSurface::PickPtag() will return same value for instance and source geo surfaces
- For surfaces that are created from replicator can have different item returned via ILxGeoCacheSurface::SourceItem() depending if source mesh used for replication is visible or not.