SDK White Paper

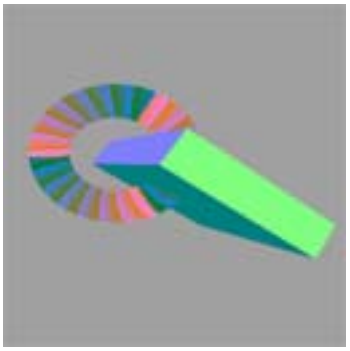
Occlusion Query
Checking for Hidden Pixels

WP-01402-001_v01
July 2004

nVSDK

Abstract

Occlusion Queries



Many graphics engines today waste time by attempting to draw pixels of objects that do not appear in a scene. These hidden objects are *occluded* from view, but often the engine rendering a scene does not know which pixels are occluded and which are visible.

Tracking information about which pixels are occluded in a scene and which are visible is not a trivial design problem. Because metadata about the relation of objects in a scene is not available, most modern hardware supports the concept of an "occlusion query." Occlusions queries ask the hardware to report on whether or not a selection of the frames rendered were occluded when the buffer is presented to the screen.

Occlusion queries provide useful information that can be used in a variety of ways to reduce the load on the graphics card. Strategies for using the information include not drawing objects that are not visible and reducing other non-graphics related loads, such as scaling down physics and AI for objects that are not visible.

The demonstration described in this document presents a technique for not drawing an extremely complex object once it becomes completely occluded.

For more information, contact

Bryan Dudash
bdudash@nvidia.com

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050

The D3D API provides an interface, called IDirect3DQuery9, which gives access to an occlusion query. Each call to IDirect3DQuery9 represents a single outstanding query. This section briefly describes the IDirect3DQuery9 API. However, please consult the DirectX9 documentation for more information.

IDirect3DQuery9 API for Occlusion Queries

HRESULT IDirect3DDevice9::CreateQuery(D3DQUERYTYPE Type, IDirect3DQuery9** ppQuery);

To make an occlusion query, pass in `D3DQUERYTYPE_OCCLUSION` as the type, and a pointer to a query interface object. From then on, the query object interface is all you'll use.

HRESULT IDirect3DQuery9::Issue(DWORD dwIssueFlags);

Wrap the render calls that you are interested in monitoring with calls to `Issue(D3DISSUE_BEGIN)` and `Issue(D3DISSUE_END)`.

HRESULT IDirect3DQuery9::GetData(void* pData, DWORD dwSize, DWORD dwGetDataFlags);

When calling `GetData()` on an occlusion query. The `pData` pointer should point to a DWORD. `dwSize` should be sizeof(DWORD). When called with no flags, the function will return immediately. It will return S_OK if the query results have returned. If not, then the query is still outstanding.

Note: You can also call with `D3DGETDATA_FLUSH` in `dwGetDataFlags`. This blocks the return of the `GetData` call until the occlusion query result has been returned. However, this approach slows your application noticeably.

ULONG IDirect3DQuery9::Release(void);

Be sure to free all objects that you create.

General Occlusion Query Usage

In general, most graphics cards buffer a few frames to allow some efficiency in the parallel work between the driver and the card. In order to use a single occlusion query, you are must frame lock your application to buffer only one frame ahead (for example, flush). Forcing the driver into lockstep with the card will lower performance.

Because each occlusion query only represents one test, use multiple queries if you want to test multiple objects in a scene.

If your simple occlusion hull does not fully cover your complex object, the object will blink in and out of the complex object.

Steps to Use an Occlusion Query

The steps to use an occlusion query are as follows:

1. Render the scene
2. Turn off color and z writes
3. Begin the query
4. Render an "invisible" simple hull object that covers complex object
5. End the query
6. Test the results of the occlusion query
7. Turn color and z writes back on
8. If coarse hull is completely occluded, then the query is done
9. If coarse hull is not occluded, then render the complex object

Note: These steps show what needs to happen for a single occlusion query. If you are testing multiple objects, maintain a queue of queries to avoid putting the hardware and driver into lock-step.

Included with this demo is a class called LatentOcclusionQueryBank. This class wraps up a queue of occlusion queries to handle the standard frame queuing that occurs on most modern graphics hardware

API

HRESULT GetLatestResults(DWORD *count);

Returns S_OK if new results are returned since the last time `GetLatestResults` was called. The occlusion query fragment count is passed back in the count parameter.

HRESULT BeginNextQuery();

Issues the beginning of a new query. Returns `S_FALSE` if it has run out of queries.

HRESULT EndNextQuery();

Issues the end of a previously begun query. Returns `S_FALSE` if it has run out of queries or if it cannot find a previous query.

unsigned int GetNumActiveQueries();

Returns the number of queries that have not yet had results delivered. This number is updated when `GetLatestResults` is called.

void Free();

Frees all occlusion queries that have been allocated. Always call `free` when you are done.

API Mapping

`LPDIRECT3DQUERY9`	→	`LatentOcclusionQueryBank`
`GetData()`	→	`GetLatestResults()`
`Issue(D3DISSUE_BEGIN)`	→	`BeginNextQuery()`
`Issue(D3DISSUE_BEGIN)`	→	`EndNextQuery()`
`Release()`	→	`Free()`

Note: Replacing all calls to a single IDirect3DQuery9 object with the equivalent calls into LatentOcclusionQueryBank should work smoothly.

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com