Technote 1109

Optimizing QuickDraw3D 1.5.3 Applications For Maximum Performance

CONTENTS

There are a number of techniques not discussed in the QuickDraw3D (QD3D) documentation which a developer can use to boost performance in a typical QD3D application. This Technote presents these techniques. This Technote assumes you are familiar with 3D graphics fundamentals and QD3D programming as described in the book 3D Graphics Programming With QuickDraw 3D by Addison-Wesley.

Introduction

3D software developers continue to add more and more powerful features to their applications. However, as these applications become more and more complex, their overall performance may suffer as well. The current QD3D documentation contains no discussion of how to improve performance in a typical QD3D application. This Technote present some techniques for doing so. Also included is a discussion of what affect 3D accelerators can have on application performance.

Techniques For Optimizing QD3D 1.5.3 Applications For Maximum Performance

There are a number of things QD3D application developers can do to get maximum performance out of their applications:

Use The TriMesh Geometry

TriMesh Performance

The trimesh geometry was first introduced with QD3D 1.5 (refer to Philip Schneider's article "New QuickDraw3D Geometries" in *develop* issue 28 for a good discussion of the various QD3D geometries, and to the document "Trimesh.pdf" on the QD3D 1.5.3 SDK for a discussion of the trimesh). With QD3D 1.5.3, performance of the trimesh geometry when used in immediate mode with the interactive renderer is quite good. If you care about speed, you'll definitely want to consider using the trimesh. Depending on your needs, many of the other QD3D geometries may work just as well - but if speed is important, consider using the trimesh. Future versions of QD3D will address performance of the other geometries too.

Also, if you are going to use the trimesh, be sure and use it correctly to achieve maximum performance. Don't try building one giant trimesh with 10 different texture attributes - that won't work (see the next section discussing materials in trimesh objects).

Use Only One Texture Per TriMesh Object

In general you'll get better performance if you use only one texture per trimesh object. The texture attribute should be in the trimesh data's `triMeshAttributeSet`. Don't attempt to apply a texture to individual triangles or vertices.

Supply A Bounding-Box For The TriMesh

The trimesh geometry contains an optional bounding-box parameter (see the `bBox` parameter in the `TQ3TriMeshData` data structure) that can be provided to accelerate culling/clipping. Do specify a bounding-box when creating the trimesh geometry and you'll get better performance (however, make sure the bounding box you specify is correct, or a crash may result).

Apply Only the Normal & UV Coordinate Vertex Attributes

Apply only the following vertex attributes to a given object if you want better performance: normal (`kQ3AttributeTypeNormal`) & UV Coordinate (`kQ3AttributeTypeSurfaceUV` & `kQ3AttributeTypeShadingUV`).

Reuse As Many Vertices As Possible

Reuse as many vertices as possible. Vertices with duplicate coordinates may have different normals meaning you'll have to make a wasteful duplicate of the vertex. In such cases, you should consider averaging the normals and removing the duplicate. This will usually result in a smoothing effect on the geometry, but it will make things go faster.

Use the Triangle/Face Attribute Face Normal

With the current implementation of QD3D version 1.5.3, you'll want to use the face normal triangle/face attribute, and even then there are cases where you might not want to do this. If you do not supply a face normal, QD3D will calculate one for you every time you render. In many cases, this will cause QD3D to slow down, since calculating a normal is slow. However, if you have a lot of data, and it's unlikely your arrays of normals stay in the L1 or L2 cache between renderings, then it can be faster to *not* provide the face normal.

A cache miss can cost more cycles than just letting QD3D recalculate the normal. In most cases, you'll want to provide the face normal, but run some tests and see if your application works better without them.

Also, it's best if you make sure the vertices for each triangle are stored in the counterclockwise direction. Refer to the document "Trimesh.pdf" on the QD3D 1.5.3 SDK for more details.

Do Your Own Object Culling

Do your own object culling? QD3D's object culling uses bounding box culling, which is fairly accurate and is really the only way to go for a general purpose library like QD3D, but it's very slow. You should implement your own object culling function which does spherical culling.

Also, use backface culling for closed objects (when appropriate), as this rendering style is likely to be significantly faster than the other backfacing styles (refer to Chapter 6, "Style Objects", of *3D Graphics Programming With QuickDraw 3D* for the details).

Minimize Other Processing While Rendering

Applications may perform many other activities while rendering, such as drawing a wireframe view of a scene into another window, updating the coordinates of a geometry, updating non-QD3D structures, etc. All these activities can have an effect on rendering. If you can reduce the amount of time spent on these activities at rendering time, you'll see a performance improvement in your application.

Minimize Time Given To Background Applications

Many applications give too much time (via the `WaitNextEvent` function) to background applications while rendering, so no matter how much QD3D acceleration is available, there are always a few ticks delay between redraws during rendering. If you can minimize the amount of time spent servicing background applications, your application will be that much faster.

Be Careful When Using Group Objects

QD3D group objects are convenient for storing and managing objects. However, many developers fail to realize each group will push and pop the graphics state at execution time because by default, the group inline flag (`kQ3DisplayGroupStateMaskIsInline`) is not set (see Chapter 10 "Group Object" of *3D Graphics Programming With QuickDraw 3D* for the details). Pushing and popping of the graphics state at execution time for each group object can affect performance in a bad way.

If you need to push and pop the graphics state manually, use the `Q3Push_Submit` and `Q3Pop_Submit` functions. Developers instead, may want to implement their own data structures to handle the management of large numbers of objects.

Retained vs. Immediate Mode

Depending upon your particular needs, using immediate mode may be faster than retained mode. For example, if you are dealing with sphere or cone geometries with the constant subdivision style set you will probably see a performance improvement when immediate mode is used.

You'll really need to experiment to see which mode works best for you. In general, if most of a model remains unchanged from frame to frame, you'll probably want to use retained mode imaging to create and draw the model. If, however, many parts of the model do change from frame to frame, you might consider using immediate mode imaging, creating and rendering of a model on a shape-by-shape basis. You can, of course, use a combination of retained and immediate mode imaging: you can create retained objects for the parts of a model that remain static and draw quickly changing objects in immediate mode. Refer to Chapter 1 of 3D Graphics Programming With QuickDraw3D for a discussion of both retained and immediate modes.

Don't Use `Q3View_Sync`, `TQ3ViewEndFrameMethod` Unless Necessary

It's best you not use the `Q3View_Sync` or `TQ3ViewEndFrameMethod` functions unless absolutely necessary when using the interactive renderer. These functions essentially put the renderer into synchronous mode. Try to avoid doing so, and you'll get better performance.

Using 3D Accelerators To Boost Performance

There's been an increase lately in the number of 3D graphics accelerator boards which support QD3D. These accelerators are designed to also improve performance for most QD3D applications. However, many developers are disappointed to find their applications don't exhibit much in the way of improved performance with these boards installed on their systems. One reason for this is many developers are not using the techniques discussed in the previous section for obtaining maximum performance with their QD3D applications.

When looking at the performance of a given 3D accelerator, it is important you understand most cards accelerate the 3D graphics rasterization process. As you know, rasterization is one process (usually towards the end) in the rendering pipeline. It follows, then, if you are using a geometry in your program which is computationally expensive, such as a NURB geometry, and your code and geometry calculations take up a great deal of time, even if a given accelerator could "instantaneously" render polygons you wouldn't see much of a speed improvement in your program.

It is also likely that at a certain point (application/geometry/complexity), an application will wind up performing at the same speed whether a software renderer or a 3D accelerator board is being used. Developers can experiment to see where exactly this tradeoff takes place.

If you make rasterization more "expensive", for example by adding high quality (tri-linear mip mapped) texturing, transparency, and/or CSG operations, then you will see a large performance and quality gain over a software renderer (of course this is very much dependent on the 3D accelerator card in use). To really take advantage of a given 3D accelerator card, you need to tailor your application to take advantage of the specific card's features.

Most cards will, with varying degrees, accelerate QD3D applications, and will often improve substantially the appearance of textures. However, it is possible to bog a card down, as many cards don't properly handle all of the complexity that general purpose applications generate in a scene.

Summary

If you follow the above guidelines, you'll see a big performance boost in your QD3D application. If you don't believe us, and you would like to see how much the above techniques help speed up your application, there is a retail product called **3DMF Optimizer** from Pangea Software (http://www.realtime.net/~pangea) which uses many of the techniques discussed in this note and let's you see the speed boosts we're talking about. A demo version of 3DMF Optimizer is available at the URL listed above.

Further References

- *3D Graphics Programming With QuickDraw 3D* , Addison-Wesley
- "New QuickDraw3D Geometries" by Philip Schneider, *develop* magazine issue 28
- **3D Optimizer**, from Pangea Software

Downloadables

Acrobat version of this Note

Technotes
Previous Technote | Contents | Next Technote