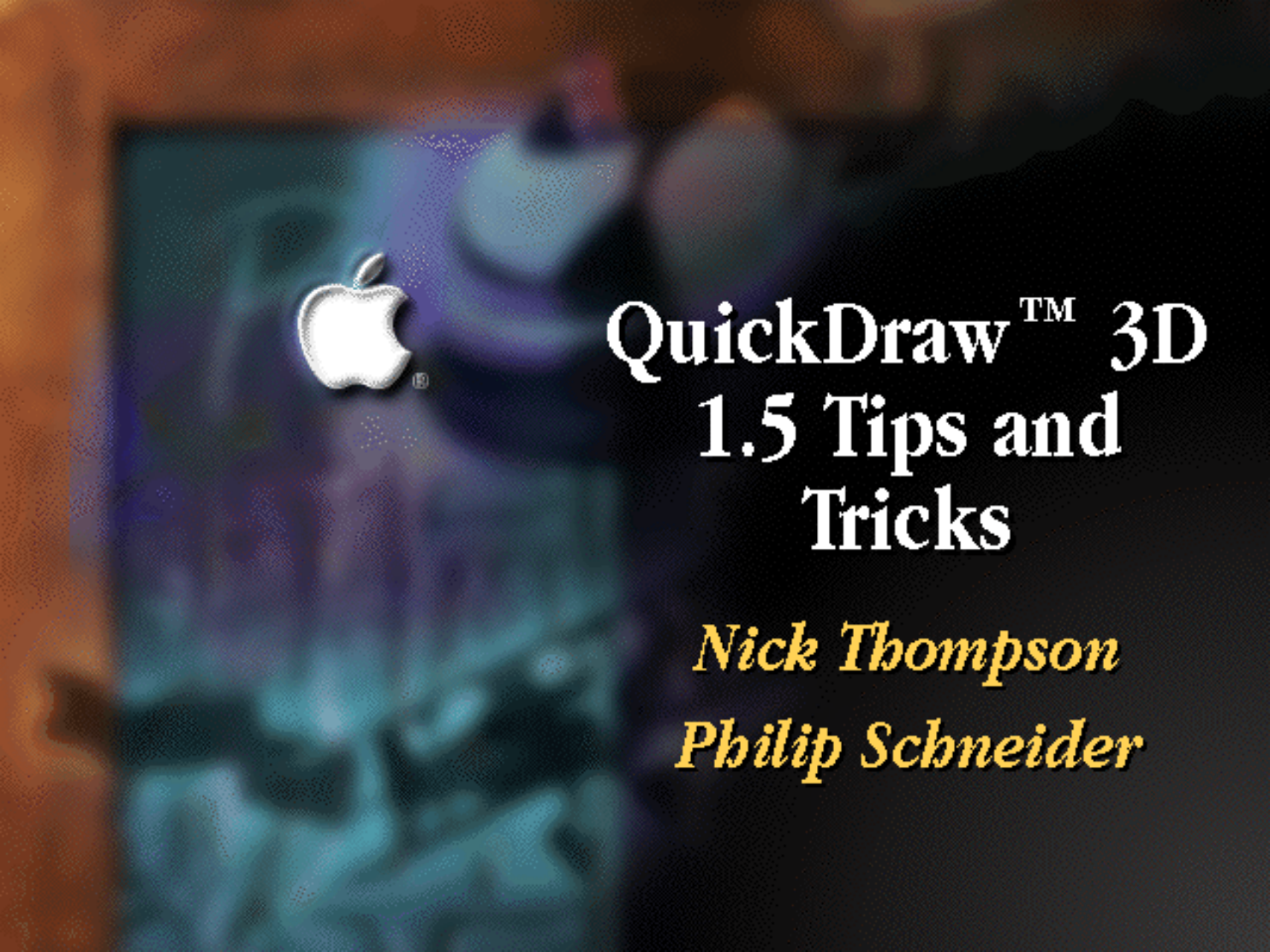
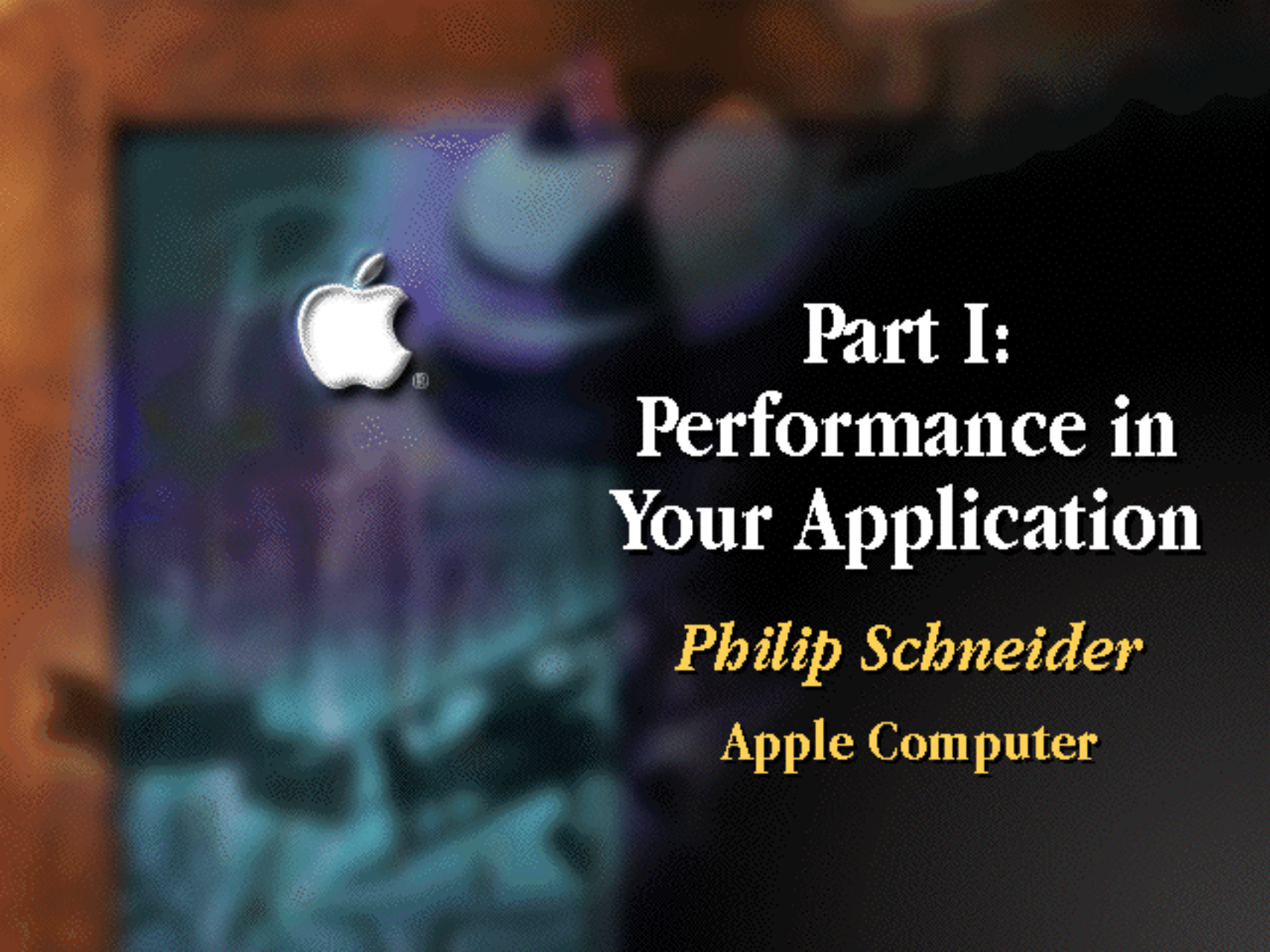

Worldwide
Developers
Conference


QuickDraw™ 3D
1.5 Tips and
Tricks
Nick Thompson
Philip Schneider

How to Use QuickDraw 3D 1.5 Effectively in Your Product

Taking advantage of 1.5 features

- Performance and tips
- Plug-in renderer support
- How to write plug-ins
- Picking


Part I:
Performance in
Your Application
Philip Schneider
Apple Computer

Performance and General Tips

Making best possible use of QuickDraw 3D 1.5

- Choosing the best geometry types for your application
- Structuring your application effectively
- Tuning
- Finding out the library version

Choosing Appropriate Geometry

Characteristic	Polyhedron	Trimesh	Mesh	Trigrid
Memory usage	very good	good	poor	very good
File space usage	very good	good	very good	very good
Rendering speed	good	very good	good	good
Topological obj editing	poor	impossible	very good	impossible
Topological data struct ed.	fair	fair	impossible no data str.	impossible fx topology
Geometric data structure editing	very good	very good	impossible	very good

Efficient Use of Groups

- Group traversal can introduce overhead
- Groups push and pop
  - The saving and resortation of state data will introduce overhead
    - It may be more efficient to implement your own data structures
    - …And/or use immediate mode
  - You can avoid this by using the kQ3DisplayGroupStateMaskIsInline state flag

Setting the In-line Flag

kQ3DisplayGroupStateMaskIsInline

- Use the API call Q3DisplayGroup_SetState
- Beware
  - If your group contains transforms or sets colors, this may not help you
  - These things set state, this may not be what you want

Making Use of Hardware Acceleration

IR automatically uses H/W, but...

- If you use a pixmap based draw context acceleration may not work
- Because many cards only accelerate Macintosh draw contexts
  - So only use pixmap draw contexts for operations that absolutely require them
    - Compositing, other post processing, and printing are the best candidates for a pixmap draw context

Other Hints and Tips

- Backface culling
  - Set this on to speed up drawing
- Number of lights
  - More lights will slow drawing down
- Quality fallbacks
  - Draw in edge mode during interactions
  - Draw flat shaded during interactions

Finding Out the Library Version

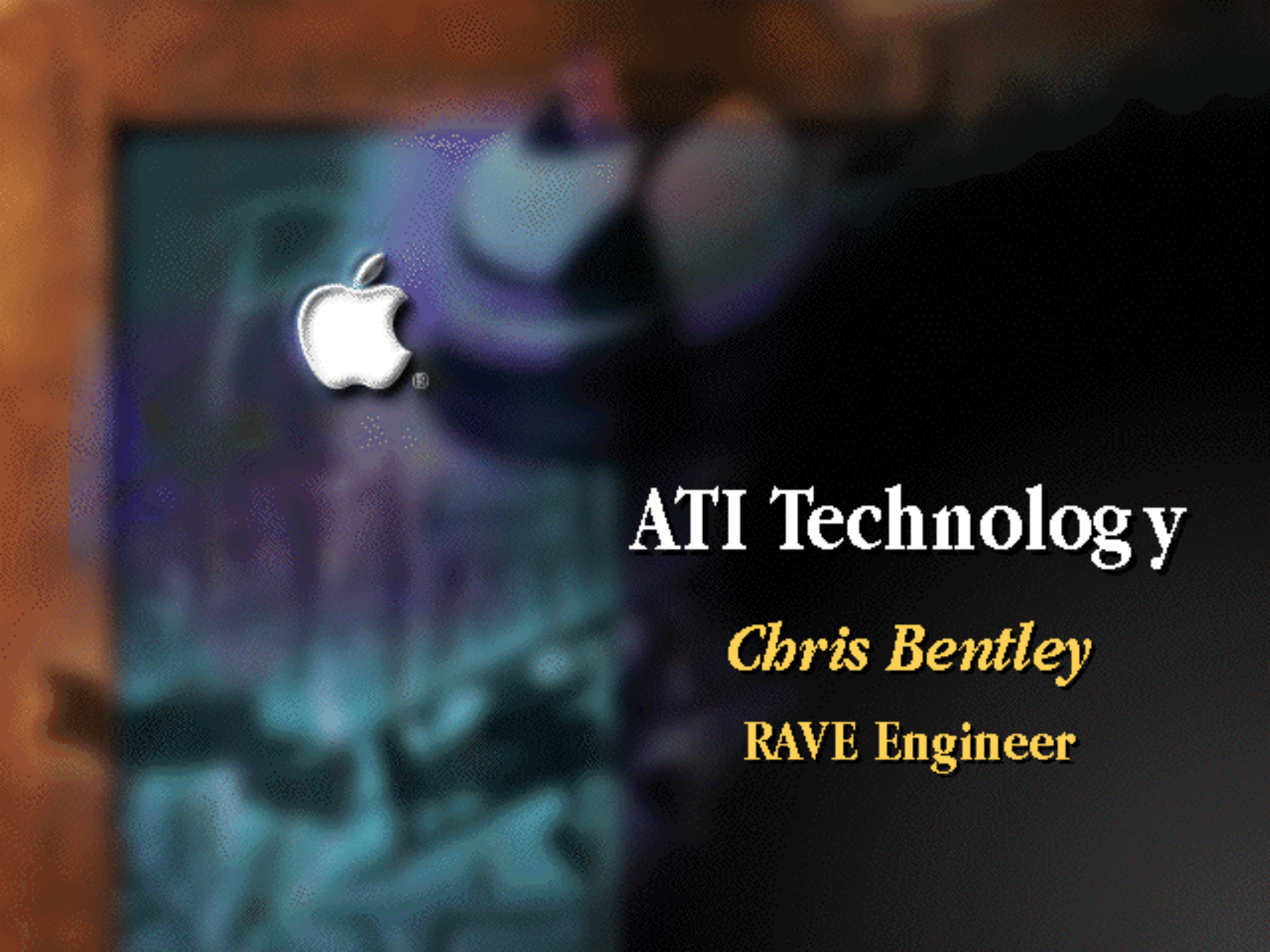
Returning the "release" versions...

- We introduced a new way to determine which version of both QuickDraw 3D and the QuickDraw 3D Viewer Lib are installed:
  - Q3ViewerGetReleaseVersion
  - Q3GetReleaseVersion
- Both return the version in a long
  - Similar to 'vers' format—e.g. 0x01518000 ==> 1.5.1 release

QuickDraw 3D RAVE

Unified access to H/W acceleration

- If your application needs high performance rendering
  - For example:
    - Games, simulations, real time applications
- You may want to take a look at RAVE


ATI Technology
Chris Bentley
RAVE Engineer

Hints and Tips for ATI

- By-pass buffer clear to boost performance
- Avoid cost of kQABufferComposite notification method
- Cache context work around for GWorlds
  - Call QADrawContextNew() with kQAContext_Cache flag
  - Register for kQAMethod_BufferComposite notification method
  - Post process rendered buffer and use CopyBits()

Notification Callback Methods

```
TQANoticeMethod noticeMethod;
noticeMethod.TQABufferNoticeMethod = cBack;

QASetNoticeMethod(
    context,
    kQAMethod_BufferComposite,
    noticeMethod,
    NULL );

void cBack(
    const TQADrawContext *drawContext,
    const TQADevice buffer,
    const TQARect *dirtyRect,
    void *refCon )
{
    /* post process, CopyBits */
}
```

More Hints and Tips

For ATI Technology's 3D RAGE

- Support for rectangular mip maps
- Add and delete textures between RenderStart() & RenderEnd()
- Looking forward... RAGE PRO
  - Strips and fans
  - VQ texture compression
  - Texture compositing
  - LOD biasing

ATI Developer Support Program

Advanced technology seeding

- Developer purchase program
  - Send email to devrel@atitech.ca
- Support - SDKs - Sample code
  - http://www.atitech.ca
- Co-marketing opportunities


Part II: Plug-in
Renderer Support
Nick Thompson
Apple Computer

Why Use Plug-in Renderers?

Adding value to your product

- Allow you to leverage the work of other renderer developers:

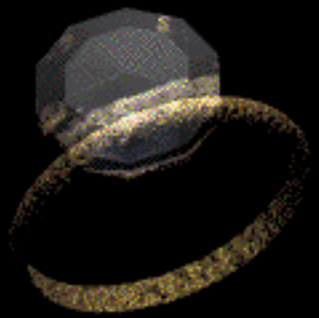


Image ©1997 LightWork Design Ltd.

Image courtesy Kevin Mathews, ©1997 Artifice

Using Plug-in Renderers

Scott Burgess

Electric Cafe

Electric Café

ModelShop 3.0 and ModelShop VR

- ModelShop VR
  - Superfast, using TriMesh and Macintosh Draw Context
- ModelShop 3.0
  - Large projects and photorealism
- Public beta of ModelShop VR at http://www.eleccafe.com

Using Plug-in Renderers in Your Product

Classification

- Plug-in renderers can be classified in two ways
  - Interactive
    - ThinkFish, Apple Interactive, Apple Wireframe
  - Non-interactive
    - LightWork Design “SuperLite” renderer
    - Ray Tracers, Per-pixel Shading renderers

Presenting an Interface

- The interface depends on the renderer type
- A renderer registers whether it is interactive or non-interactive
- You can determine if the renderer is interactive using
  - Q3Renderer_IsInteractive(renderer) ;

UI Suggestions

- If the renderer is interactive add it to a renderer menu
- If the renderer is non interactive
  - Provide a mechanism to render the data into a new window
  - Or a window that the user is not able to interact with

UI Suggestions

- If the user can manipulate the view
  - Use an interactive renderer for that view
- Use an interactive view to set up orientation
  - Then render into a separate window using a non-interactive renderer
  - … or write to file
  - … or disallow manipulation
    - Fall back to last used interactive renderer if the user clicks in the window

Finding Out Which Renderers Are Installed

- Query the object system using
  - Q3ObjectHierarchy_GetSubClassData() ;
  - Pass in object type you want to query for:
    - In this case “kQ3SharedTypeRenderer”
  - Pass in a struct of type TQ3SubClassData
- Free up the TQ3SubClassData using
  - Q3ObjectHierarchy_EmptySubClassData

Traversing the Sub-class Data—**very* simplified!!*

```
Q3ObjectHierarchy_GetSubClassData(
    kQ3SharedTypeRenderer, &subClassData);

classPointer = subClassData.classTypes;

for( i = 0; i < subClassData.numClasses; i++ ) {

    if( *classPointer != kQ3RendererTypeGeneric ) {
        Q3RendererClass_GetNickNameString(
         *classPointer, objectClassString );

        if( objectClassString[0] == '\0' ) {
         /*renderer didn't provide name,use class
name*/
          Q3ObjectHierarchy_GetStringFromType(

    *classPointer, objectClassName);
        }
        /* use the string for whatever */
    }
    classPointer++ ;
}
Q3ObjectHierarchy_EmptySubClassData( &subClassData ) ;
```

Getting the Name of the Renderer

- Renderer Nicknames
  - Used for user interface elements
    - Menus, dialogs etc.
  - Are localizable
  - Available from 1.5.1 onwards
    - Use Q3RendererClass_GetNickNameString
    - Check the library is version 1.5.1 before calling this function

Renderer Preferences

Allow you to customize the renderer settings

- Allow you to adjust the options for a renderer
- You can query the current renderer
  - Use Q3Renderer_HasModalConfigure
    - To determine if it has a preferences dialog

Renderer Preferences

Example with LightWorks SuperLight renderer

```
/* Put up the configure dialog */
if (Q3Renderer_HasModalConfigure(qd3dRenderer))
{   /* enables a movable modal with event handler
*/
    qd3dAnchor.clientEventHandler = HandleEvent ;
    qd3dStatus =
        Q3Renderer_ModalConfigure
          qd3dRenderer,
          qd3dAnchor,
          &qd3dCanceled);
}
```

Use the code above to enable the prefs dialog

Your HandleEvent proc is called to handle events

Renderer Preferences

- If you pass NULL in the clientEventHandler field
  - You get a modal prefs dialog
- If you pass in an event handler
  - You get moveable modal dialog
  - Remember to:
    - Disable all but edit menu and Apple menu items
    - Event handler is only called if plug-in doesn't handle the event

Saving and Restoring Renderer Preferences

- Q3Renderer_GetConfigurationData
  - Gets private renderer configuration data
    - Which can be saved in a preference
    - Applications should tag this data with the Renderer's object name.
- Q3Renderer_SetConfigurationData
  - Use this to restore renderer settings

Plug-in Renderers

Dair Grant

LightWork Design Ltd

LightWork Design SuperLite

- SuperLite renderer is on sale
  - Macintosh and Windows
- Developers should provide support for plug-in renderers in their applications
- Bundling for SuperLite is available
- Developers can also license higher end versions
- http://www.lightwork.com

Part III—Plug-in Basics

How to write plug-ins for QuickDraw 3D

Plug-in Basics

- Plug-ins can be used to extend the functionality of Quickdraw 3D
- In 1.5 the following types of plug-ins are supported
  - Elements/Attributes
  - Groups
  - Renderers
- Support for plug-in shaders is planned for a future release

Commonality

Features common to all plug-ins

- Loading/Initialization
  - On Mac OS this is handled by CFM
  - On Windows by the DLL loader
  - You need to supply a registration function
- Metahandler
  - Method dispatcher for the plug-in class
- Termination

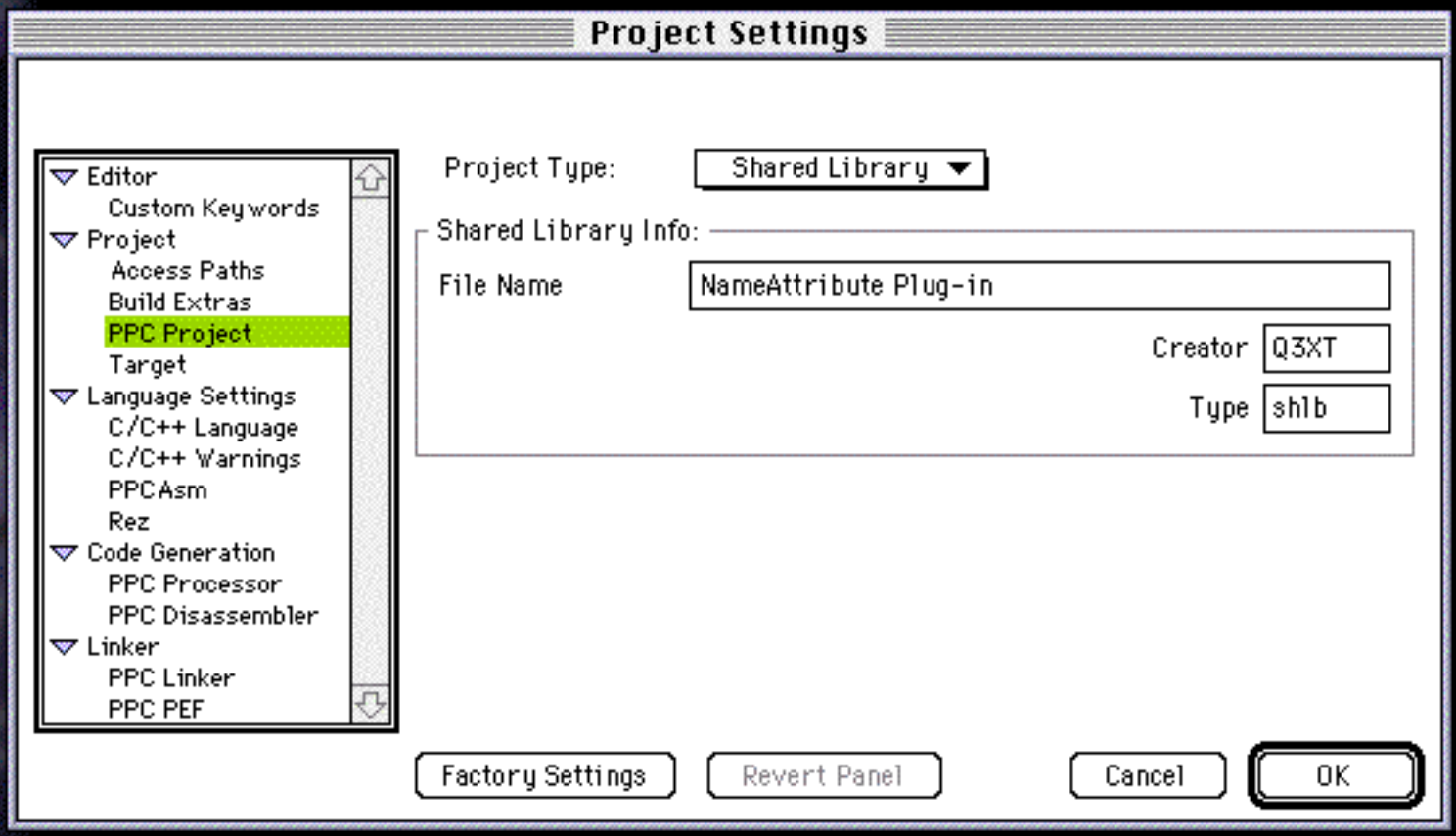
Loading and Initialization

Setting up your plug-in

- For Mac OS
  - Packaged as a CFM shared library
    - Creator 'Q3XT' type 'shlb'
- For Windows
  - The file extension must be .Q3X
- Entry points and exit points map to functions in the shared library

Setting Up a Plug-in

CodeWarrior Project Settings Dialog



Mac OS Initialization Function

Gets called by CFM

- You supply a registration function that is called later when QD3D loads the extension

```
/* Name attribute CFM init routine */
OSErr NameAttribute_ConnectionInitializationRoutine(
                        InitBlockPtr initBlock)
{
    TQ3XSharedLibraryInfo sharedLibraryInfo;

    sharedLibraryInfo.registerFunction = NameAttribute_Regist
    sharedLibraryInfo.sharedLibrary =
                (unsigned long)initBlock->connectionID;
    Q3XSharedLibrary_Register(&sharedLibraryInfo);

    pSharedLibrary = (unsigned long)initBlock->connectionID;
    return noErr;
}
```

Windows Initialization Function

- Similar to Macintosh, but both initialization and termination are handled in the DLL Main entry point for the library

Summary of Initialization and Loading

- Register the shared lib with QuickDraw 3D
- QuickDraw 3D handles loading of library
- Register function
  - In the example this was NameAttribute_Register()
  - QuickDraw 3D calls this function when it is ready to initialize your plug-in

Metahandler

The plug-in method dispatcher

- Each class of plug-in has a set of methods that get called via the metahandler
- These methods vary depending on whether your plug-in class is
  - Element, Attribute, Group, or Renderer
- The method handler is essentially a big switch statement
  - Returning (usually) function pointers based on the set of constants passed in

Metahandler

Example

```
static TQ3XFunctionPointer NameAttribute_MetaHandler(
    TQ3XMethodType     methodType)
{
    switch (methodType) {
        case kQ3XMethodTypeObjectClassVersion :
            return (TQ3XFunctionPointer)
                        Q3_OBJECT_CLASS_VERSION(
                            majorVersion, minorVersion);
        case kQ3XMethodTypeObjectTraverse :
            return (TQ3XFunctionPointer)
                        NameAttribute_Traverse;
        /* a bunch of other method dispatchers ... */
        default:
            return (TQ3XFunctionPointer) NULL;
    }
}
```

Metahandler

Notes

- The example shows three things:
  - Versioning
  - Returning a method pointer
  - Default behavior
- There are handlers that must be defined for each specific type of plug-in
  - See the documentation for details

Versioning

You should supply a plug-in version

- CFM will be used to decide which of identical plug-ins to use
  - Based on the fragment name
- Supply a version
  - So that QuickDraw 3D is able to discern the correct version for your plug-in
  - If no version is supplied, the default is to set the version to “0.0”
  - Only the highest version will be loaded

Returning a Method Pointer

- Check the header files for the format of a function associated with a particular constant
- For example:

```
#define kQ3XMethodTypeObjectTraverse
        Q3_METHOD_TYPE('t','r','v','s') /* byte count */

/*
 *  TQ3XObjectTraverseMethod
 *
 *  The "data" is a pointer to your internal element data
 *
 *  The view is the current traversal view.
 */
typedef TQ3Status (QD3D_CALLBACK *TQ3XObjectTraverseMethod
    TQ3Object      object,
    void           *data,
    TQ3ViewObject  view);
```

Returning a Method Pointer

- Irrespective of the format of the return value, it is always cast to type TQ3XFunctionPointer
- Some constants return a value NOT a function pointer
  - Version (kQ3XMethodTypeObjectClassVersion)
  - IsDrawable (kQ3XMethodTypeObjectIsDrawable)

Default Behavior

- In the case where you don't supply a method, return NULL
- If appropriate the default method gets called if you return NULL

Termination on Mac OS

Called by CFM

- On Windows the DLL Main function gets called with DLL_PROCESS_DETACH
- On Mac OS you need to supply this:

```
void NameAttribute_ConnectionTerminationRoutine (void)
{
    TQ3Status theStatus ;

    if( pSharedLibrary != NULL ) {
        Q3XSharedLibrary_Unregister(pSharedLibrary);
        pSharedLibrary = NULL;
    }

    theStatus = NameAttribute_Unregister() ;
}
```

Elements and Attributes

Introduction

- Used to add custom data to QD3D objects
- See *develop* Issue 26
- What's the difference between them
  - Attributes can be inherited, elements are not
- There is a sample plug in attribute on the conference CD

Registration

Called by QuickDraw 3D

- This is the function registered earlier

```
TQ3Status   NameAttribute_Register( void )
{
    TQ3ElementType
    myElementType = kElementTypeName ;


    pNameAttributeClass =
        Q3XElementClass_Register(
            &myElementType,
            kElementTypeNameString,
            sizeof(TQ3StringObject),
            NameAttribute_MetaHandler );


    if (pNameAttributeClass == NULL)
        return kQ3Failure;

    return kQ3Success;
}
```

Registration

Important note

- Registration is by name
  - The binary type for your class is assigned at runtime, and returned to you in the registration function
    - Use this returned type in calls like Q3Set_Add()
      - Either save a reference to the type returned by the register call
      - Or use Q3XObjectClass_GetType to get the type back from the system

Attribute/Element Methods

- For a description of the methods an attribute can implemented see pages 6 through 7 of "Adding Custom Data to QuickDraw 3D Objects" in *develop* issue 26
- A copy of this is on the conference CD

Name Space Changes

Changes from develop and the Book

- If you are reading the documentation for attributes note the following changes (there are more than this, but this will give you the general idea)

```
TQ3FunctionPointer              --->  TQ3XFunctionPointer
TQ3MethodType                   --->  TQ3XMethodType
Q3View_SubmitWriteData          --->  Q3XView_SubmitWriteData
kQ3MethodTypeObjectTraverse     --->  kQ3XMethodTypeObjectTraverse
kQ3MethodTypeObjectReadData     --->  kQ3XMethodTypeObjectReadData
kQ3MethodTypeElementCopyAdd     --->  kQ3XMethodTypeElementCopyAdd
kQ3MethodTypeElementCopyGet     --->  kQ3XMethodTypeElementCopyGet
```

Registering a Plug-in Group

This applies to plug-in renderers too

- First define your unique object type by using “Q3_OBJECT_TYPE” macro
- Declare a data structure to store private data for your plug-in

```
#define kQ3XXXGroup
    Q3_OBJECT_TYPE('X','X','X','G')

typedef struct XXXGroupPrivate{
    // XXX Private Data
} XXXGroupPrivate;

TQ3ObjectClass XXXGroupClass;
```

Registering a Plug-in Group

- Different registration API call than for Elements and Attributes

```
TQ3Status XXXGroup_Register(void)
{
    XXXGroupClass =
    Q3ObjectHierarchy_RegisterClass(
        kQ3GroupTypeDisplay,     // Parent Type
        kQ3XXXGroup,             // Group Type
        "XXXGroup",              // Group Name
        XXXGroup_MetaHandler,    // MetaHandler
        NULL,                    // VirtualMetaHandler
        0,                       // Methods Size
        sizeof(XXXGroupPrivate));// Instance Size

    if (XXXGroupClass == NULL)
        return kQ3Failure;

    return kQ3Success;
}
```

Plug-in Examples and Documentation

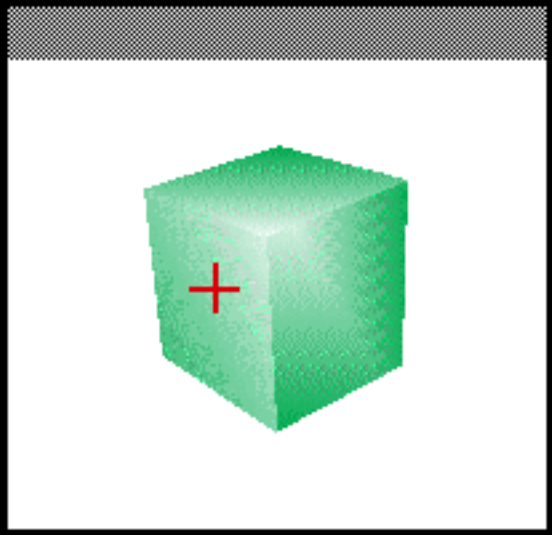
- On the SDK and the WWDC CD
  - Plug-in attribute: ‘name’
  - 2 Plug-in renderers (Simple renderer and Wireframe)
  - Plug-in group: display proxy group (DPG)
    - This is a simple level of detail group

Part IV—Picking With 1.5

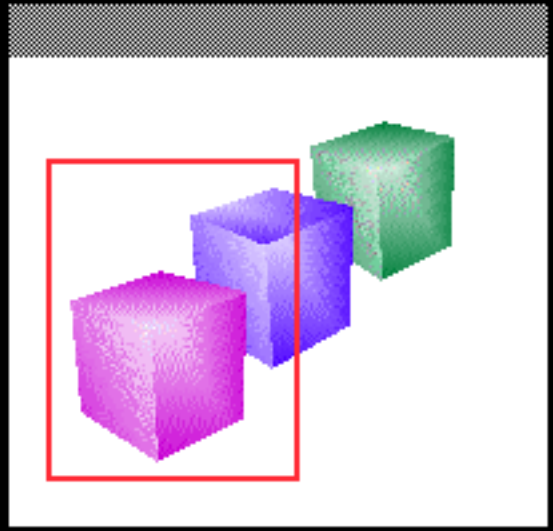
Managing user selection with QuickDraw 3D

Types of Pick Objects

Window Point



Window Rectangle



Kinds of Pick Detail Information Calculated…

- TQ3PickDetail specifies information calculated per hit:
  - Pick ID
  - Group hierarchy path
  - Reference to object hit
  - Local to world matrix

Kinds of Pick Detail Information Calculated

- TQ3PickDetail *(cont)*
  - XYZ intersection point
  - Distance from camera
  - Surface normal vector
  - Shape part
  - Surface UV parameterization

Geometry Shape Pick Parts

TQ3PickParts style specifies parts tested

- Object Level
  - Pick intersects a geometry only once anywhere
- Part Level
  - Pick intersects several parts of the same geometry
  - Object / Face / Edge / Vertex or any combination

Picking Process

- 1. Setup and create a pick object
- 2. Submit objects in a picking submit loop
- 3. For each hit
  - Get pick detail information
  - Use this information for interaction
- 4. Specify a new pick location
  - (Repeat steps 2 through 4)
- 5. Dispose the pick object

Pick Setup Information

- Choose type of pick object
- Initial pick location
- Pick detail information
- Sorting method
  - Near to Far / Far to Near / None
- Maximum number of hits
- Vertex and edge tolerances

Picking Submit Loop Example

```
TQ3Status SubmitPickObjects(TQ3ViewObject view,
                            TQ3PickObject pick,
                            TQ3GroupObject group,
                            TQ3StyleObject subDivStyle)
{
    TQ3ViewStatus viewStatus;

    Q3View_StartPicking(view, pick);
    do {
        Q3Style_Submit(subDivStyle, view);
        Q3DisplayGroup_Submit(group, view);
        viewStatus = Q3View_EndPicking(view);
    } while (viewStatus == kQ3ViewStatusRetraverse);

    return (viewStatus == kQ3ViewStatusDone) ?
            kQ3Success : kQ3Failure;
}
```

Pick Hit List

- Hits accumulate in the pick object and are retrieved after performing a submit loop
- A hit contains the pick detail information for a single intersection
- The maximum number of hits to be returned are specified at setup
- Hits are referenced by index
- Hits are sorted relative to distance from the viewer

Querying a Pick for Hits Example

Get intersection point in world space

```
TQ3Status GetWorldPoint(       TQ3PickObject   pick,
                        TQ3Point3D    *worldPoint)
{   TQ3Status          status;
    unsigned long      numHits;

    Q3Pick_GetNumHits(pick, &numHits);

    if (numHits == 0)
        return kQ3Failure;

    status = Q3Pick_GetPickDetailData(
                pick,
                0,
                kQ3PickDetailMaskXYZ,
                worldPoint);
    return status;
}
```

Querying a Pick for Mesh Shape Parts

```
TQ3Status ChangeMeshPart(TQ3PickObject  pick)
{
    TQ3ShapePartObject  shapePart;
    TQ3MeshComponent    comp;

    status = Q3Pick_GetPickDetailData(
                pick,
                0,
                kQ3PickDetailMaskShapePart,
                &shapePart);

    if (shapePart == NULL  || status == kQ3Failure)
        return kQ3Failure;

    status = Q3MeshPart_GetComponent(shapePart, &comp);
    …
```

Querying a Pick for Mesh Shape Parts

```
    …
    switch (Q3MeshPart_GetType(shapePart)) {
        case kQ3MeshPartTypeMeshFacePart:
            DoFace(component,shapePart);    break;

        case kQ3MeshPartTypeMeshEdgePart:
            DoEdge(component,shapePart);    break;

        case kQ3MeshPartTypeMeshVertexPart:
            DoVertex(component,shapePart); break;
    }
    return kQ3Success;
}
```

Summary—The Three Ps

- Performance
  - Making the right design decisions
  - Consider writing to RAVE
- Plug-in support
  - Developer opportunity providing plug-ins
  - Makes your application more desirable for your customers
    - By adding valuable features that YOU DON'T HAVE TO IMPLEMENT YOURSELF!!
- Picking


Q&A