Geometric Model Reconstruction from Streams of DirectX 3D Game Application

Zhigeng Pan  
State Key Lab of CAD&CG, Zhejiang University  
Hangzhou, 310027, P.R. China  
zgpan@cad.zju.edu.cn

Xiaochao Wei  
State Key Lab of CAD&CG, Zhejiang University  
Hangzhou, 310027, P.R. China  
weixch@cad.zju.edu.cn

Jian Yang  
Centrality Communications Corp., Shanghai, 201206 P.R. China  
jyang@centralitycomm.com.cn

ABSTRACT
This paper presents a method of intercepting data stream, command stream and rendering states from DirectX 9 graphics pipeline by hooking the low-level graphics library. It also shows that it is useful to reconstruct geometric models or render models with new styles. The paper shows not only how the basic mechanics of intercepting streams in the DirectX 9 graphics pipeline lead to a non-invasive extension mechanism for graphics applications, but also how to manipulate the streams and states of pipeline properly to reconstruct geometric information and export models of different styles. While describing how our system efficiently reconstructs a declarative representation of the geometry implicit in the graphics library command stream, a set of application extensions built with this framework is presented including the replayer and reconstructer.

Categories and Subject Descriptors
I.3.1 [Computer Graphics]: Hardware Architecture (B.4.2); I.3.4 [Computer Graphics]: Graphics Utilities

General Terms
Algorithms

Keywords
Reconstruction, Geometric Model, DirectX 9, Graphics Pipeline.

1. INTRODUCTION
Non-invasive methods work by intercepting calls to the underlying graphics library, which is one of the possible attacks in the graphics pipeline [4]. In this way, a variety of useful changes were allowed, such as changing the visual style of interactive graphics programs. These methods, which alter the behavior of graphics applications, can be used to extend graphics applications without modifying the applications and make it be possible for adding new functionality to existing applications. While intercepting DirectX 3D, there is a trade off between performance and the cost for low level streams containing less required information. Thus the state of the graphics pipeline should be recorded efficiently. In this paper, a state machine based solution is introduced to record the states and the recorded states are exported to corresponding script later. In addition, data structures representing the relation between geometric elements are also recorded for handling of the intercepted geometric data.

1.1 Interception of Library
A graphics application can be intercepted in various levels. When intercepting higher-level scene graph APIs, such as OpenInventor, it would severely limit the information obtained from the stream. Also, in this case, the purpose of intercepting library would be limited in fewer situations. However, intercepting at a much lower level is not a wise choice, either. For example, considering the pixels going to the frame buffer, they are formed in a very unstructured way; intercepting them requires expensive computer vision analysis to create any structures for further processing. Examples of this approach are the painterly rendering systems of [5, 9]. Most low-level graphics APIs, including OpenGL, do not draw the whole scene in the frame at a time. The command stream represents a programmatic or imperative representation of the geometry: it is an instruction list; when executed, the list would drive graphics pipeline for enough times to draw a whole picture of the geometry in the frame. In contrast, most descriptions of geometry for geometric processing algorithms, and high-level APIs (such as Renderman [16] or OpenInventor [15]) represent geometry declaratively as triangle meshes with connectivity or subdivision surfaces.

1.2 Intercepting OpenGL
As to OpenGL graphics library, many existing systems intercept it for various purposes. SGI’s glTrace [10] records the sequence of library calls for debugging assistance, performance monitoring and hardware timing simulation. IBM’s ZAPdb OpenGL debugger uses this interception technique to aid in debugging OpenGL applications. Intel’s Graphics Performance Toolkit [8] also uses the similar method to optimize graphics application performance. In 1996, SGI developed a stream codec for OpenGL [14]. This feature made it possible to capture and record the related streams into files. While these streams are saved, the software can examine and
process it off-line, however, the toolkit does not mean to add new functionality and change the application in non-invasive way. WireGL [6] and its successor Chromium [7] also intercept OpenGL to generate different outputs like distributed displays. Chromium provides a mechanism for implementing plug-in modules that alter the stream of GL commands. HijackGL [1, 2] also intercepts the OpenGL stream. This permitted making localized changes to the stream. To combat the limitations of these localized changes, some of the HijackGL renderers buffered an entire frame of data and processed it to render the image. In a sense, these renderers were performing a special-case of the scene reconstruction.

(a) The scene intercepted from 3DMark03 (b) The scene model without texture (c) The scene model with texture (d) The monster model with texture

Figure 1: Resonstructed models

1.3 Intercepting DirectX 3D
In contrast to OpenGL, DirectX 3D is more complicated in architecture and difficult to be intercepted. The most common toolkits of this category are Fraps [3] and Unamed Direct3D Benchmarking Tools [11]. Fraps is a universal Windows application that can be used with all games using DirectX or OpenGL technology. Two game examples from our group [13, 12] are based on OpenGL and DirectX respectively. It monitors the call to OpenGL APIs and DirectX interfaces. While playing 3D games, Fraps will inspect the frame buffer for video capture and FPS test. However, it can only replay the video file captured from frame buffer rather than rendering process, so it is unavoidable to induce errors when testing different graphics cards. Unlike Fraps, Unamed Direct3D Benchmarking Tools intercept the calls to DirectX interface. They save the vertexstream, rendering commands into files at the end of game. The replayer will replay the course of rendering by loading data form files recorded. Since the replayer directly loads the stream recorded in files without extra software computations on vertex and texture, which is executed in game, the BenchFPS value is higher than RecFPS of Fraps. Avoiding the error in testing, it has a higher requirement to memory size and disk capacity for the three files. Since DirectX is based on COM, it is necessary to define image classes derived from the DirectX interfaces. When game application invokes the DirectX library to create instance of interface, our library intercepts the command and invokes the function in real DirectX library to create the real instance of interface maintained by our library. Then, our library will return an instance of corresponding image class representing the real instance for the game application.

(a) A rough model obtained (b) A rough model with texture (c) A tiger of original style (d) A tiger of grass style

Figure 2: Models of different styles

2. ARCHITECTURE OF THE SOFTWARE
This paper explores the potential of reconstructing original models and rendering models with new effects (see Figure 1 and Figure 2) in interactive 3D applications, using limited information in the graphics library command stream. After examining the information obtained by intercepting the command stream of the low-level graphics library of DirectX 3D, directly manipulating it can lead to some simple stylistic changes. Our software contains six modules: intercept module, trace module, record module, replay module, debug module and reconstruct module (see Figure 3).

1. Intercept module is used to monitor the invoking to the real DirectX graphics library. The module intercepts the vertex/index stream, texture stream, command stream. The intercepted streams are transferred to the following modules for later manipulations.
2. Trace module uses the stream transferred from intercept module and outputs the commands invoked by game application in HTML style.
3. Record module classifies the stream into pipeline state machine, resource map and then saves them to files or transfers the structures to the following modules.
Figure 3: Framework of our software

4. Replay module loads the files saved by record module and replay the rendering process.

5. Debug module is used to edit the command stream and states recorded in replaying files so as to eliminate bugs found in replaying process step by step.

6. Reconstruct module first parses the pipeline state machine and resource map transferred from record module, then builds models from structures and basic geometric information and saves the models to X-files.

3. RECORDING STREAMS

As shown in the Figure 3, recording is the vital and central part of the whole software. The data structures defined in the recording stage will affect the flexibility of the whole software except trace module. In order to make it more convenient for the later stages, two data structures are used: state machine and resource map. State machine is introduced to represent the pipeline in real time. Resource map is used to locate resource in memory and describe its parameters (texture category, mip level and color format, etc.). Game application renders the scene by invoking the D3D library. The instruction intercepted can be stored as command sequence, but it will produce too much redundancy. Although it can simplify the code in replay module, it will reduce the efficiency and speed of replay module and make it harder to manipulate the data in later period. DirectX graphics pipeline can be regarded as a state machine. This fact makes it possible to represent the graphics pipeline in predefined state machine.

DirectX pipeline has many states to be set before it is started by invoking instructions of drawing primitive, but in fact only a few of them are necessary to be set before rendering. After all, a frame of the most complicated scene is not necessary to use all the techniques of DirectX. For that reason, when it comes to save the state machine into scripts for replaying, it is not necessary to save all the states of the state machine. To reduce the redundancy, a delta log is introduced to only record the state changes. When replaying in series, replayer will only load the corresponding delta log into state machine to speed up replayer, and drive the graphics pipeline to rebuild the rendering process.

However, for example, when skipping from frame 1 to frame 6, the replayer must load another intact state script again instead of the delta log of frame 6. Because the delta log has to compare the state machines of two rendering processes to find the states changed, it will cost more time in the record stage.

When a scene is rendered frame-by-frame, many resources are created by game and passed into the graphics pipeline. These resources must be organized for recording and reconstructing at later stage. A common way is to classify and put them into queues for various resources, but it is not very convenient for searching by the memory address it is created when rendering. Searching by memory address is used in following modules all the time. In order to solve this problem, resource map is used. The key of the map is the memory address of the resource; the value of the map is a structure recording its parameter. All the elements in the map are classified and linked into various queues representing various resources created in rendering. When searching by address, the corresponding element can be easily found in the map. When searching in a certain kind of resource category, the element can be found by beginning searching from the head of the queue.

4. MODEL RECONSTRUCTION

When it comes to the reconstruct stage, only the resource description in resource map is not enough. In this stage, x model files should be exported. The data contained in the resource used by pipeline must be obtained in real time and computed for advanced geometric information. Without the basic and advanced geometric information, the generated models would apparently be bags of disjoint triangles. Instead, because of computing connectivity, data obtained at early stage may be converted to any standard mesh boundary representation.

4.1 Reconstructing Models

In DirectX graphics library, vertices have several attributes including location, color, normal, and texture coordinate which can be set using function pair “lock” and “unlock” on VertexBuffer interface. When this command is issued, all of a vertex’s attributes are fixed.

DirectX graphics pipeline does keep track of vertices even after rasterization of the primitives associated with it. The resource interface set into the pipeline can be tracked and the data could be obtained by locking the interface. But when intercepting the library, the interface description stored in pipeline state machine can be used to fetch the memory address of the interface for the rendering efficiency of game. In this way, it is well proved that the current resource pipeline can be obtained by efficiently cutting off the cost of invoking “get” function.

Before each drawing is invoked the pipeline state machine will export the position data to vertex_mem from current vertex streams. From current vertex streams, data can also
be obtained for normal_mem, TexCoord_Mem. According to the primitive type and index buffer, if there is one, in state machine, the triangles are assembled in Index_mem. Since it is hard to export the vertex blend effect to model file, only one world transform matrix is necessary to record for a rendering. Because the view/project matrices are only used in rendering, they could be ignored when reconstructing models.

4.2 Exporting Models

Figure 1 demonstrates the results of our geometry capture module. This extension works by processing the reconstructed geometric information and exporting as an X-File of the scene data. After a 3D scene is captured, the model can be viewed and manipulated by other software such as Deep Explore.

In the X-File, the texture can be exported, and then be changed to realize another effect. However, some special effects are not possible to be added into the model file, such as vertex blend and multi-texture. In this software, the shaders are recorded and saved, but they are not imported into the model file to give the special effects. Chromium and its predecessor, WireGL [13] exist primarily to generate different output. Their system, like ours, uses library interception techniques. It sends OpenGL commands to a cluster of machines that render to a tiled display for large format output. However, our software reconstructs models using the value saved in state machine and geometric info structure with many of the redundant values in all the streams removed.

4.3 Conclusions and Future Work

In this paper we developed a system to intercept DirectX 3D graphics library and reconstruct geometric models from intercepted streams.

In the future, we will improve the system to make it not only export model files but also affect files. Effects are a higher-level abstraction that can be compiled into object files. The object file can then be used to create an effect with D3DXVECTOR3. Thus other 3D modeling software such as Maya can use the reconstructed models directly.

5. ACKNOWLEDGMENTS

This project is co-supported by 973 project (grant no: 2002CB312100), Excellent Youth Teacher Program of MOE in China. We would like to thank Mr. Bolin Yang and Chengxiang Zhang for their kind help on this project.

6. ADDITIONAL AUTHORS

Additional authors: Shusen Sun (State Key Lab of CAD&CG, Zhejiang University,Hangzhou, 310027, P.R. China, email: ss@cad.zju.edu.cn) and Wen Yan (State Key Lab of CAD &CG, Zhejiang University,Hangzhou, 310027, P.R. China, email: yanwen@cad.zju.edu.cn).

7. REFERENCES


