RC22613 (W0210-193) October 29, 2002
Computer Science

IBM Research Report

Pervasive 3D Viewing for Product Data Management

Bruce D'Amora, Fausto Bernardini
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Pervasive 3D Viewing for Product Data Management

Bruce D'Amora and Fausto Bernardini

Abstract

A 3D viewer designed for mechanical CAD application scenarios is presented. We designed the viewer for low memory and processor speed handheld devices such as the Pocket PC. In this paper we describe the motivation for such an application. We developed a software 3D rendering pipeline to support the application. We elaborate on the design decisions made to maximize performance and image quality on resource constrained devices. We describe implementation details that should facilitate the readers understanding of the performance issues resolved. Finally, we include thoughts about future technical developments needed to extend the usefulness of this application within Product Life cycle Management (PLM) and other application segments.

Keywords

Product Data Management (PDM), Product Life cycle Managment (PLM), Mechanical CAD (MCAD),Personal Digital Assistant (PDA)

Introduction

Over the last several years there have been significant advances in mobile computing platforms. Handheld devices such as cellular phones and Personal Digital Assistants (PDAs) have become practically ubiquitous. With innovations in processor, power management, and wireless technology, handheld platforms will be able to support a greater variety of applications.

As PDAs become smaller, more powerful, and less expensive, applications previously reserved for workstation, desktop, and laptop systems are becoming available on these more mobile compute platforms. Existing applications are not only being modified to accommodate the resource constraints of handheld devices, but new applications are being designed specifically to meet the mobility requirements of a PDA user. One industry segment that provides an opportunity for such applications is manufacturing.

Mechanical Computer-aided Design (MCAD) applications traditionally focus on the design and analysis phases of the production process. These applications typically require powerful workstations with large storage systems and high performance 3D graphics adapters. Other solutions to this problem include server side rendering and transmission of video streams to the PDA via wireless as discussed by Woodward et al [1]. We have provided a solution for users who require access to MCAD 3D models during stages of the manufacturing process where use of design workstations is impractical or limited. In this scenario, a user at a remote manufacturing location requires access to an MCAD drawing of a specific part. The user can reference this part with the PDA 3D Viewer and annotate with customer feedback or speculative design changes *(Figure 1*). The transfer of data from a parts database can be accomplished via wireless or direct synchronization. Unlike other handheld 3D Viewing software, e.g. ParallelGraphics Cortona™, AutoCAD PocketCAD™, our system utilizes a server side application during download to transcode the VRML representation of an MCAD 3D model to a highly optimized data format appropriate to PDA devices. The capability of interaction with and annotation of a digital model, allows the user to capture more meaningful feedback than is possible without a visual reference. The necessity to orient the models to specific views is invaluable in describing a modification to a part. Saving and

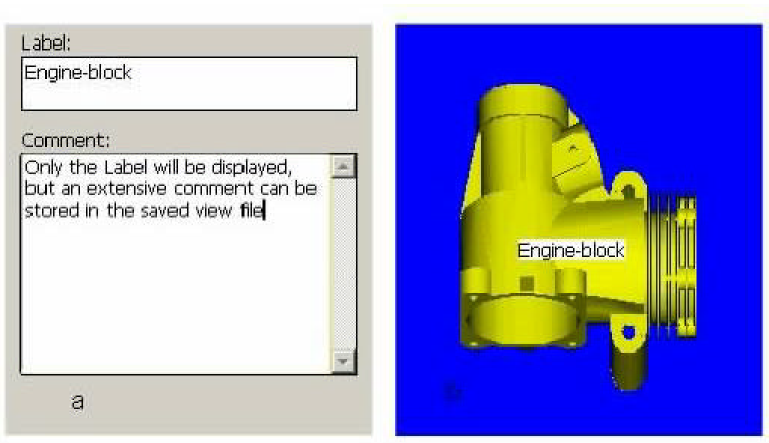
Label:
Engine-block
Comment:
Only the Label will be displayed, but an extensive comment can be stored in the saved view file

Figure 1 a.) text dialog b.) annotated model restoring annotations with specific 3D model views provides a mechanism to capture multiple views and annotations in a single file that can be transferred to the server for access by the engineer/designer. Another challenge was to define a useful set of rendering capabilities and interactions for an MCAD application and develop it for a resource constrained device without sacrificing performance or reducing image quality. Also, when designing such an application, it is crucial to adequately address restrictions in user interface specifically in the areas of user input, i.e. lack of keyboard and mouse/tablet devices. In the remainder of this paper, we will attempt to demonstrate that with certain design decisions and careful implementation, an application for mobile users requiring 3D MCAD viewing and annotation capabilities can perform well on resource constrained handheld devices.

Architecture

The 3D viewer we have prototyped is written in C/C++ and is optimized to execute on Pocket PC handheld devices. We created a Transcoder that converts VRML 2.0 files into a proprietary binary format. The Transcoder compresses geometric and image data such as textures [2] to minimize storage requirements for 3D models on the PDA device. The viewer parses transcoded VRML files and applies geometric transformations and scan conversion to render triangles and lines to a memory color buffer which is then copied to the

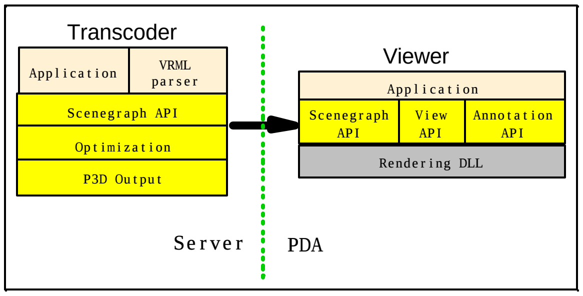

Figure 2 Architecture

devices framebuffer *(Figure 2)*. The Transcoder includes an API so it can be integrated into an application or run as a stand alone application on a server. We have added user interface the enables the user to manipulate the on screen models via the stylus or PDA buttons.

In an effort to not only enable a platform for PDM applications, but to provide general API for addressing various 3D application segments, we have defined and implemented a set of low-level 3D functions which have been packaged into a shared library written entirely in C. We have also included an API packaged as a shared library for creating and maintaining a scene graph representing the graphics database constructed from the transcoded VRML data file. The graph is composed of transform and shape nodes. Each shape node contains references to indexed face sets, normal and texture coordinate buffers. We have included an additional appearance field in each shape node that contains information about textures, material properties, and lighting. During traversal of the scene graph, function entry points in the low-level 3D API are called to set rendering state and geometry . All data is passed via pointers to avoid excessive duplication and

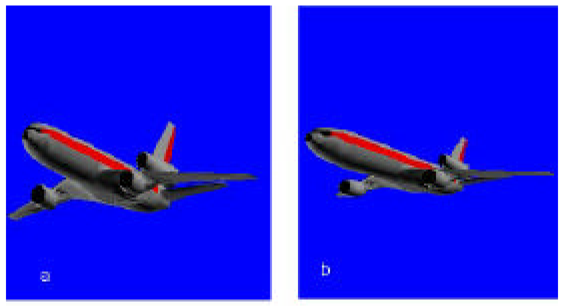

Figure 3 a.) Orthographic b.) Perspective

processing of data via extraneous copy operations. The inset shows an example of the API for the low-level rendering library as well as the scene graph.

The rendering pipeline renders lit, textured, and Gouraud-shaded triangles and lines. Geometry is projected via orthographic or perspective projection *(Figure 3)*. The renderer supports multiple attribute binding types: *color per primitive, color per face, color per vertex, color per corner, normal per face, normal per vertex, normal per corner, texture per vertex, and texture per corner*. *Figure 4* shows some of the various attribute bindings supported. We have implemented this via "specialty" rasterizers that are loaded based on changes in the attribute state of the geometry being rendered. The state machine approach has been used before in other 3D APIs such as OpenGL [3]. Although this increases code size of libraries - a potential impact to model storage on PDAs - the performance increase from eliminating multiple branches within inner loops justifies this design decision. In an effort to minimize floating point operations which are particularly slow on Pocket PC devices that employ emulation libraries, we have minimized the amount of floating point in the rendering pipeline. Specifically, we have only maintained individual transformation matrices in floating point until they are concatenated at which

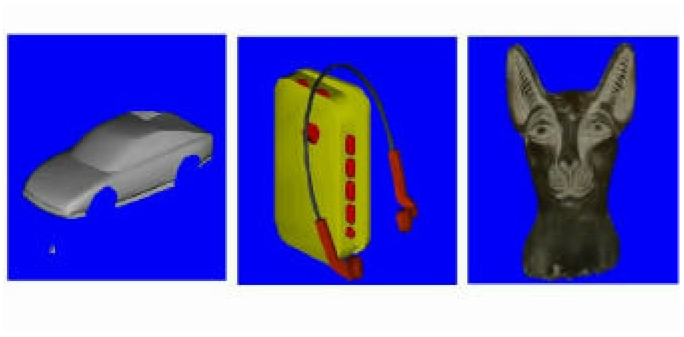

Figure 4 a.) normal per vertex b.) color per corner c.) texture per corner

time they are converted to short integers. The vertex, color, and texture coordinate data is stored and maintained as short or unsigned short integer throughout the rendering process. The Phong lighting equation is applied for directional diffuse lighting and forcing symmetrical view scaling allows us to update lighting by applying the transpose of the rotation matrix to the light vector [4]. In order to provide functionality particularly useful to the MCAD viewing applications, We have included annotation, save/restore view, and a select capability. The goal was to include support for display and interaction with digital mockups while at a remote customer location. . We added function that allows the user to select a specific part or area of the drawing and create a short text label to be displayed with the drawing as well as a more detailed comment . This textual information along with the current view of the 3D model can be stored in a binary file for later viewing or transfer to a server via internet or USB synchronization. We have provided a class library for adding, deleting, modifying, saving, and restoring this information. User interaction is affected by the lack of a 2 or 3 button mouse on PDAs. We chose to address this difference with desktop platforms by designating screen areas for viewing transformations such as rotation, scaling, and translation.

Implementation

Geometry Pipeline

In creating the prototype viewer, we encountered several technical and usability challenges. First and foremost was the memory sizes of the PDA devices. Pocket PC devices typically include 32 or 64MB of memory for storage of data and applications. This can be a challenge for storage of MCAD drawings that typically represent multiple parts. We alleviated this problem by creating the Transcoder that converted VRML files into a compact binary representation using signed and unsigned short integer data types rather than floating point data. The use of short integer also reflects the need to eliminate as much floating point computation as possible since native floating point support is not available on the Pocket PC processors. The geometry pipeline maintains separate matrices for model/view and projection transforms in floating point, but converts the final concatenated transformation to short integer before applying to vertex buffers. This was done to minimize transformation error and maximize performance. A more significant problem that we found to directly effect rendering performance was the size of the instruction and data cache. The Pocket PC processors we tested on had 16KB instruction and data caches. Vertex data included x, y, z, r, g, b values each being represented with 16-bit signed integer totaling 12 bytes per vertex. Geometry buffers with more than 1,365 vertices would overflow the data cache (assuming that no local data was being stored). 3D MCAD models often contain hundreds of thousands of triangles which would present a problem when processing data buffers. In order to accommodate this cache memory limitation, we organized our data into buffers that fit in the cache and organized our code to render all the triangles in a buffer before processing the next buffer.

Rasterization

Pocket PC devices currently have no 3D acceleration. It was necessary to create an optimized triangle scan converter. Performance was the primary goal. Rather than use a parallel polygon rasterization algorithm [5] more commonly found in current hardware implementations, we decided on a scan line approach. This approach has the advantage of only processing pixels within the boundary edges of the triangle. Also, parallel polygon rasterization algorithms are optimal for multiple rendering pipeline architectures. We were utilizing a single threaded uniprocessor platform. We used a modified Bresenham [6] algorithm to compute polygon edge intersections [7]. This approach was incremental and minimized divides to one per edge. Sub-pixel accuracy was achieved by shift scaling all vertex attributes during geometry processing and then re-scaling before using in color, texture, or Z-buffer lookups. We avoided inner loop branching by creating fragment processors for each attribute binding type: *color per primitive, color per face, color per vertex, color*

Low-level Rendering API

```
void Transform::draw() const
{
   ::RenPushMatrix();
   ::RenTranslate(translate.x, translate.y, translate.z);
   ::RenTranslate(invcenter.x, invcenter.y, invcenter.z);
   ::RenRotate(rotat.x, rotate.y, rotate.z, rotate.angle);
   ::RenRotate(invscalerotate.x, invscalerotate.y,
                              invscalerotate.z, invscalerotate.angle);
   ::RenScale(scale.x, scale.y, scale.z);
   ::RenRotate(scalerotate.x, scalerotate.y, scalerotate.z,
        scalerotate.angle);
   ::RenTranslate(center.x, center.y, center.z);

  // set accumulated rotation for lighting transform
   Matrix4 r = Matrix4::rotation(rotate, rotate.angle);
  ::RenMultRotationMatrix(r.getFloat());
  Group::draw();
   ::RenPopMatrix();
}
```

Scene graph API

```
Void Node::AddTransformNode()
BoundingSphere* sphere = new BoundingSphere(cv, radius);
transform->setBoundingSphere(sphere);

Transform->setTranslate(tx, ty, tz);
Transform->setCenter(cx, cy, cz)
Transform->setRotate(r_axis, r_angle);
Transform->setScale(sx, sy, sz);
Transform->setScaleOrientation(so_axis, so_angle);
transform->setChildCount(numchildren);
```

per corner, normal per face, normal per vertex, normal per corner, texture per vertex, and texture per corner.

Results

We are able to achieve polygon rendering rates of 200K triangles/sec for unlit, untextured models. If lighting is enabled performance decreases on average by 25%. Performance bottlenecks differ depending on several factors including the vertex attribute binding and the size of the triangles being rendered. For small triangles the overall rendering rate is limited by transformation of vertices and triangle setup. For larger triangles or textured triangles, the overall rendering rate is limited by interpolation of colors or texture coordinates. *Figure 5* shows the performance improvements from the initial raster engine to the current version. The raster performance is characterized by time per stage per frame. The model used during this analysis was a 3000 triangle object with an average triangle area of 5 pixels. The raster engine was broken into 5 stages - *triangle setup, edge setup, edge walk, and interpolation*. The performance impact of cache misses due to repeated loading/unloading of the color framebuffer was also measured. The results demonstrate that all stages except cache miss performance were improved resulting in approximately a 2x overall rasterization performance improvement. The increase in cache miss time may be due to optimizations in the other stages that resulted in the cache being more fully utilized than in the initial implementation increasing the probability of a cache miss from a color framebuffer access. The improvements to the initial rasterizer were on both the algorithmic and implementation level. The initial rasterizer attempted to compensate for precision loss of interpolants by maintaining separate integer and fractional variables. Subsequently, this was changed to using a scaled integer approach.

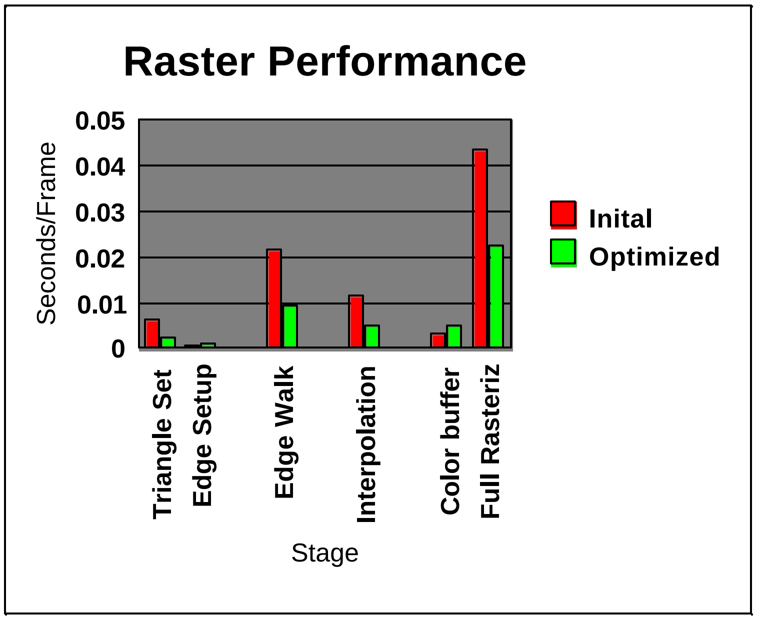

Figure 5 Raster Performance

Conclusions

As handheld devices become more powerful, they will become a viable alternative platform for traditional desktop applications as well as new mobile applications. Over 13.1M PDA units were shipped worldwide in 2001 up 18% from 2000.[1] It is also predictable that the processor speeds will increase as the chip manufacturers discover innovative ways to decrease power requirements and fabrication costs to meet the demands of the increasing mobile and embedded device market. As this happens, we expect to see corresponding performance improvements in our software renderer which are predominantly integer based and should scale with clock speed.

Another factor is the availability of low-power 2D and 3D graphics chips. Certainly, the trend is for chip providers to develop low power full function 3D chips targeted for the mobile laptop market. ATI 2D video chips have recently been used in the Toshiba e740 Pocket PC™. What will drive these manufacturers to target lower power handheld devices? One enabling application may be gaming. The $9.35B U.S. Video game industry[2] is being driven by hardware platforms such as Nintendo GameCube™, Sony Playstation™, and most recently Microsoft Xbox™. Another significant phenomena is the popularity of the handheld Nintendo Gameboy Color™ and Gameboy Advance™. Over 20 million of these devices have been sold since inception with the bulk of the revenue coming from the game software itself. These devices are very similar to PDAs with the GameBoy Advance even utilizing the same ARM™ processor core as many PocketPC models.

As other user interfaces develop specifically voice recognition and motion tracking, handheld devices will have additional opportunity to be utilized as tools in maintenance and support training.

The handheld market will continue to grow driven by increased demand for mobile computing and supported by the technology trend of low-cost powerful processors. This growth will be accelerated as traditional desktop applications adapt themselves to this low-cost platform and new mobile applications become available.

References

1. Woodward,Charles and Valli,Seppo and Honkamaa, Petri and Hakkarainen,Mika, Wireless 3D CAD Viewing on a PDA Device, Proceedings of the 2nd Asian Mobile Computing Conference 2002

2. Balmelli, Laurent and Bernardini, Fausto and Taubin, Gabriel, Space-Optimized Texture Maps, Computer Graphics Forum, volume 21(3), pp. 411-420, 2002 (Proceeding of Eurographics)

3. Segal, M and Akeley K. The OpenGL Graphics System: A Specification

4. Moller, T and Haines, E. Real-time Rendering

5. Pineda, Juan. A Parallel Algorithm for Polygon Rasterization. SIGGRAPH 1988 Proceedings

6. Bresenham, J. Algorithm for Computer Control of Digital Plotter. IBM Systems Journal 4,1 (1965), 25-30

7. Fleisher, K and Salesin, D. Accurate Polygon Scan Conversion using Half-Open Intervals. Graphics Gems IV, p. 362

[1] Gartner Dataquest

[2] The NPD Group, Inc.