

Artist-Centric New HMI Software Development Workflow: Development of Real-Time 3D Rendering Engine for Reconfigurable Instrument Clusters

Kazuyuki Nakata, Kentaro Kajita and Ryoichi Nishikawa
DENSO Corporation

Soju Matsumoto, Takumi Ishikawa and Yoma Watarai
3D Incorporated

ABSTRACT

Instrument clusters that display all information on a TFT-LCD screen, also known as reconfigurable instrument clusters, have become the new trend in automotive interiors. DENSO mass-produced the world's first reconfigurable instrument cluster in 2008. To satisfy customer requirements, large quantities of resources were required. Coupled with an iterative process due to requirement changes, development costs became very high. Reducing development costs was vital in order to expand the reconfigurable instrument cluster products line. One solution was to use existing human machine interface (HMI) tools. However, most HMI tools are geared toward software developers and not graphic artists. Furthermore, each tool has its own unique method for image and scene creation, creating an ineffective and sometimes difficult environment for artists familiar with industry-leading computer graphics (CG) software to learn and use the tools. As a result, the artist-to-target workflow suffered from inefficiencies because software developers had to remodel the artists designs created with the CG software. Each time the design was changed, both artists and developers had to reiterate the development process.

The instrument cluster is not only a primary source of information to the driver, but also connects the driver to the rest of the vehicle, establishing a somewhat personal relationship between human and machine. It is imperative that the instrument cluster design reflect clarity, precision and beauty. An artist-centric approach was proposed to reduce the development effort by introducing a data converter and real-time 3D rendering engine. The goal was to achieve minimal loss in the development cycle, from concept to production. The artist's idea, as expressed with industry-leading CG software (Autodesk® 3ds Max®), must be precisely replicated at the target level. The new proposed workflow was as followed: First, an artist created a 3D model and scene in 3D CG software. Next, the 3D model and scene were exported and converted for use in the rendering engine. The converted data included parameters and shader code for the rendering engine. Finally, the data was ran on the rendering engine. The rendering engine called the appropriate OpenGL ES commands based on the converted data from the artist and dynamic input data from the developer's application code. With the new workflow, any changes the artist makes are immediately and seamlessly applied to the target.

It is important that the rendering engine interprets the converted data on the embedded platform as close as possible to the artist's intended design. Using automatic shader code generation and frame rate improvement techniques contributed to rendering a high quality image on a TFT-LCD screen. The data converter was able to generate and optimize the shader code automatically based on parameters set by the artist in the 3D CG software.

The new workflow was evaluated during a mass production development project and resulted in improved software development efficiency. While DENSO has succeeded in automating graphics development with data conversion and a rendering engine, the next goal is to integrate a model-based development tool to improve development efficiency.

CITATION: Nakata, K., Kajita, K., Nishikawa, R., Matsumoto, S. et al., "Artist-Centric New HMI Software Development Workflow: Development of Real-Time 3D Rendering Engine for Reconfigurable Instrument Clusters," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 6(2):2013, doi:10.4271/2013-01-0425.

INTRODUCTION

Reconfigurable instrument clusters have become a new trend in automotive interiors, especially in the luxury cars market. Instrument clusters with small size information displays and some mechanical gauges have also become popular. DENSO mass-produced the world's first reconfigurable instrument cluster in 2008. It is a product using DVGA (1280 × 480) high resolution 12.3 inch TFT display.

One main problem with reconfigurable instrument clusters is hardware limitation, as evidenced in 2008 when only 16 MB of ROM could be afforded. Concerns over the reliability of HDD or NAND flash have prevented implementation of the memory into the instrument clusters. The limitation of memory resources is strict.

When developing instrument clusters with a small size display, animation was created by bit block transfer (bitblt). This required indication parts to be replicated as image data. Thus, to reduce file size, image compression and clipping image techniques were used to update only the parts of the image that changed during animation. However, the techniques were not enough to reduce the file size for large, high resolution displays.

For example, [Figure 1](#) shows the concept when creating the opening movie (1280 × 480 pixel, 20 fps, 5.2 sec) by bitblt. If the entire file size of the bitmap data (2.4 MB per frame) was left uncompressed for all 104 frames, the total size of the whole movie would be 256 MB. After applying an image clipping technique, the total size was still 178 MB, well above the ROM size.

To achieve drastic data reduction, OpenGL was used. OpenGL is a cross-language, multi-platform API for rendering 2D and 3D CG. It is supported by most of the new embedded graphics processing units (GPU) by default. By using OpenGL, 3D CG can be directly drawn from vertex information in high speed. This is known as real time rendering. OpenGL allows the opening movie file size to be reduced to 3 MB, well below the ROM size, while maintaining image quality as bitblt animation.

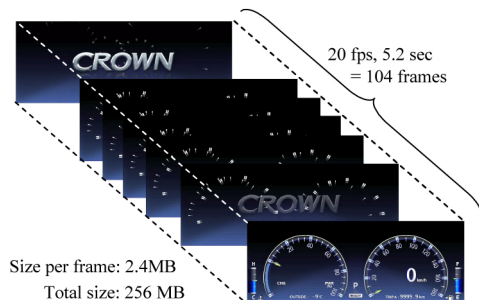


Figure 1. Implement size of animation by bitblt

One of the benefits of OpenGL is high portability. Since the API is a common standard the same behavior can be reproduced on other OpenGL hardware by porting the existing software. For example, it was easy to shift from rapid

prototyping on PC to the actual target hardware seamlessly, even after hardware changes.

During the course of development, it was evident that development cost would be a problem. To satisfy customer requirements, large quantities of resources were required. Coupled with an iterative process due to requirement changes, development costs became very high. So why is the iterative process necessary? Customer demands such as appealing high quality graphics differ between each vehicle model and may be difficult to meet due to hardware limitation. It was necessary to find a compromise by checking the rendering result on the actual target with our customers. With the increase in the flexibility of reconfigurable instrument clusters and increase in the number of functions available on each vehicle, the rapid increasing of development scale and complexity was inevitable. To pursue the product at a higher level with the customer, a need to promote development efficiency and better handle the iterative process became imperative.

ANALYSIS OF DEVELOPMENT COST

Reducing development costs was vital in order to expand the reconfigurable instrument cluster product line. To achieve the expansion, development costs had to be reduced to the same level as conventional instrument clusters. Presently, the cost of instrument clusters without a display is 40 man-months and instrument clusters with a small display is 80-160 man-months. However, the total development cost of the reconfigurable instrument cluster in 2008 was 400 man-months. The goal was to reduce the cost of reconfigurable clusters by 120 man-months, which is the same level as the instrument cluster with a small display.

Looking at the breakdown of the development costs ([Figure 2](#)), the development of graphics application held the highest cost. The main reason of the high cost is from developing graphics application code by hand and from rework from design changes by the customer. Costs could be reduced if graphics application code could be generated automatically.

In 2008, there were 3 large-scale design changes, and every time the product was recreated again. Without the ability to check the look and feel of the product until it was developed, feedback was late and the product returned to the requirement definition stage and was reworked. By utilizing rapid prototyping, the look and feel of the design could be checked during PC simulation at the stage of requirement definition and analysis, avoiding a rework on design and implementation. As a result, development costs could be reduced.

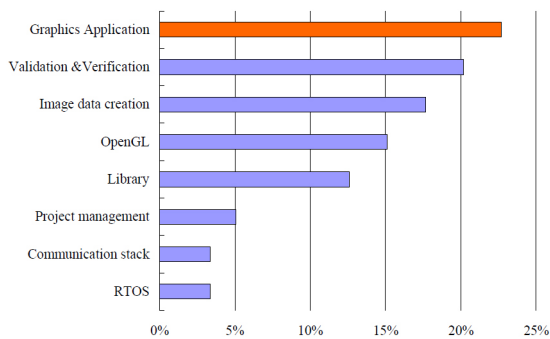


Figure 2. Breakdown of the development costs

ISSUES IN HMI TOOLS

One solution to promote development efficiency is to use existing HMI tools. HMI tools support graphical HMI development. Functions such as graphical content creation and code generation could be targeted to any number of embedded platforms. A majority of the tools also have the capability of rapid prototyping, simulation and document generation [1]. There are many HMI tools available, but none have become the de facto standard. One reason is that most of them are geared toward software developers and not graphic artists. Furthermore, each tool has its own unique method for image and scene creation, creating an ineffective and sometimes difficult environment for artists familiar with industry-leading CG software to learn and use the tools. There are tools that can import design data (for example, data created by Adobe® Photoshop®), but in many cases, it is necessary to do additional work on the data after the data is imported. With 3D model data, many tools support model importing, but scene composition is needed afterward and each tool has their own unique way of doing so.

As a result, the artist-to-target workflow suffered from inefficiencies because software developers had to remodel the artists designs created with the tools. Each time the design was changed, both artists and developers had to reiterate the development process.

ARTIST-CENTRIC NEWHMI SOFTWARE DEVELOPMENT WORKFLOW

The instrument cluster is not only a primary source of information to the driver, but also connects the driver to the rest of the vehicle, establishing a somewhat personal relationship between human and machine. It is imperative that the instrument cluster design reflect clarity, precision and beauty. An artist-centric approach was proposed to reduce the development effort by introducing a data converter and real-time 3D rendering engine. The goal was to achieve minimal loss in the development cycle, from concept to production. The artist's idea, as expressed with industry-leading CG

software (Autodesk® 3ds Max®), must be precisely replicated at the target level.

Concept

Figure 3 shows the conventional workflow. CG artist creates the CG scene, software developer recreates the data and implements the graphics application. After that, we can check the rendering result on the actual target and give feedback to the artist. So, if we found a problem by checking the rendering result, both CG artist and software developer need to rework. By this workflow, software developer's rework is needed even if it is the problem of the CG artist's work. Thus, it becomes inefficient.

Figure 4 shows the artist-centric new workflow. In this workflow, the software developer's work is replaced by automatically converting the CG scene data created by the CG artist using the data converter, exporter and preprocessor. Of course, to achieve accurate graphics and enough performance, it is necessary for the CG artist to create the data by considering the limitation and rules of the data converter and the performance of the actual target. The converted data can then be executed on the rendering engine, enabling a check of the artist's work on the actual target. If there is a problem, only the CG artist needs to rework and revise the CG scene data.

Using the previewer, the rendering result can be checked on the PC. The previewer can be used as a rapid prototype tool and give feedback to the CG artist in the early stage of the development. There are many tools that can import the CG data, but most don't support scene data such as animation. The characteristic of the new workflow is that we can utilize the work product of the CG artist directly. Details about the data converter and real time 3D rendering engine will be discussed later.

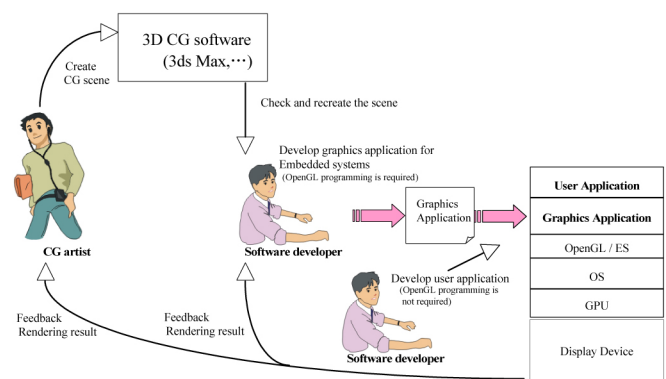


Figure 3. Conventional workflow

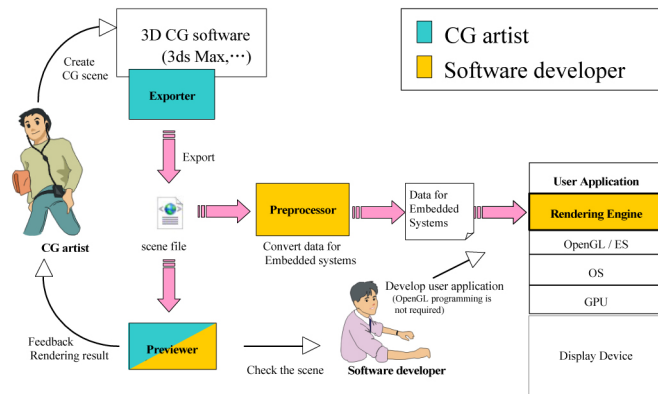


Figure 4. Artist-centric new workflow

Workflow in Depth

Figure 5 shows the work process, components and dataflow between components from scene modeling to rendering.

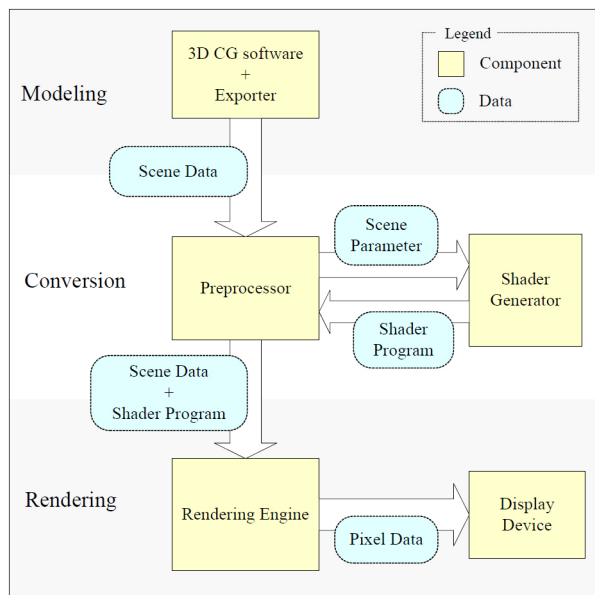


Figure 5. Overview of the work process, components and dataflow

The detail of the new workflow was as followed:

First, an artist created a 3D model and scene in 3D CG software. (Modeling stage)

Next, the 3D model and scene were exported and converted for use in the rendering engine. The converted data included parameters and shader code for the rendering engine. (Conversion stage)

Finally, the data is executed in the rendering engine. The rendering engine called the appropriate OpenGL ES commands based on the converted data from the artist and dynamic input data from the developer's application code. (Rendering stage)

Data Convertor

The data convertor consists of an exporter and a preprocessor. The exporter is a plug-in for 3D CG software and outputs the scene data, which is environment independent. The preprocessor converts the scene data for the actual target environment.

The scene data, which is the output of the modeling stage, doesn't include the shader program required in the rendering stage. Thus, output from the conversion stage was not only the converted scene data but also the shader program needed for the scene.

A shader generator module generates the shader program at the conversion stage (see Figure 5). The input for this module was the scene parameter, mainly information of the object's material or textures and light sources that light up the objects. The output of this module was source code written in shading language (GLSL) and the information to execute it.

Real-time 3D Rendering Engine

The rendering engine used the scene data generated from the conversion stage and renders to the display device. When rendering a specified object, the shader program linked to the object is used. The Rendering engine calculates the shader parameter based on information generated by the shader generator and the present status of the scene. Then the rendering engine called the graphics API to set the parameter and control the GPU.

The rendering engine should satisfy needs such as fast rendering, light-weight portability and high scalability.

Role of Software Developer

A software developer's role is only to control the animation. There are 2 types of animation, pre-defined animation and programmable animation. Pre-defined animation can be created with 3D CG software and the rendering frames for the animation can be specified directly from the user application. With programmable animation, parameters can be set for each object. For example, to render a motion for the needle of a speedometer, a CG artist creates a 3D animation that moves the needle with a constant speed from the minimum to maximum value frame by frame. The software developer would only need to write the application code that calls the rendering engine API and specify the animation drawing frame corresponding to the current value. The animation for the needle from 0 km/h to 180 km/h, is from frame 0 to frame 180. When rendering 100 km/h, we only need to call the rendering engine API with the parameter for frame number 100. OpenGL programming is not required.

Benefit

With a new workflow, any changes the artist may make can immediately and seamlessly be applied to the actual target.

In addition, rapid prototyping is available by using the previewer which could render on the PC based on the output

of the modeling stage. The point was CG artists could emulate the rendering results of the actual embedded system on a PC very easily since the previewer was specialized for the actual target environment.

Issues

It is important that the rendering engine interpret the converted data on the embedded platform as close as possible to the artist's intended design.

Using automatic shader code generation and frame rate improvement techniques contributed to rendering a high quality image on a TFT-LCD screen.

AUTOMATIC PROGRAMMABLE SHADER CODE GENERATION

As a method to maintain high image quality, the automatic shader code generation technique was introduced. A shader influences what the viewer will see by calculating the rendering effects on graphics hardware. The data converter can generate and optimize the shader code automatically based on parameters the artist will adjust from the 3D CG software. The point was shader code could be generated directly from the scene setting configured in Autodesk® 3ds Max®. Shaders are used to program the GPU programmable rendering pipeline, which allowed not only common geometry transformation and pixel-shading functions but also customized effects. The official OpenGL and OpenGL ES shading language was OpenGL Shading Language (GLSL). Shader generator can be used to generate and statically optimize the shader code automatically based on the parameter of the Blinn-Phong shading model. The Blinn-Phong shading model remains one of the more popular shading models, which produced realistic visuals with an inexpensive processing load, making it suitable for real time rendering. For example, on Autodesk® 3ds Max®, the default setting of the standard material is Blinn-Phong shading.

Shading language is defined as a programming language, allowing a programmer to code by hand. However, due to the flexibility of the programmable shader, a high level of understanding and technique would be needed to achieve a desired rendering result. By generating the shader code automatically, the creation process of the CG scene data can be simplified and development costs can be reduced for a high quality 3D CG system. The education cost for a CG artist can also be reduced.

Concept

Shader Generator Module

Shader generator module generates the shader program by the procedure shown in Figure 6.

Shader code was generated from the original template source code. To instantiate the shader code, several template parameters are needed. To decide the parameter, the scene

parameter first needed to be analyzed to collect necessary information.

Next, based on the collected information, a suitable template source code is selected and template parameters are generated at the same time. Finally, the template source code is instantiated by using the generated template parameters. The instantiation was then executed against each shader unit type.

Next, generated shader code was optimized by the optimization module. The detail of optimization will be discussed later.

Finally, the shader program information generated was used by the generated shader code on the rendering engine. The information included the required vertex elements and parameters, calculation method of the parameters, the blending method of polygons and background image.

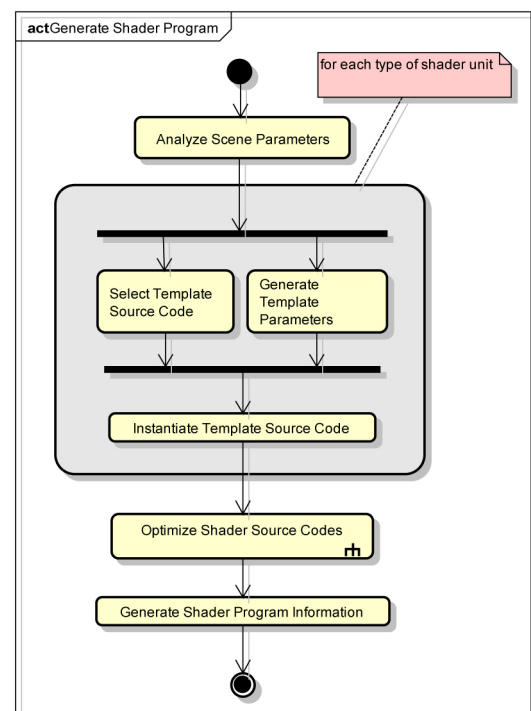


Figure 6. Procedure of shader generation

Optimization of the Shader Source Code

Optimization of the shader code has benefits:

- Generate shader code with complex calculations
- Improved rendering speed
- Initialization time of the application can be shortened

There is a limited quantity of GPU resources (for example, the number of registers and instructions) that can be used by a single shader program. With optimization, resources required for a GPU can be reduced. This allows complex calculations to be utilized by generated shader code.

Reasons for improved rendering speed: