Whitepaper

How 3D Graphics Lower System Costs for Automotive User Interfaces

Version:
1.0

Released:
6-May-2015

Purpose:
This whitepaper discusses some of the primary reasons why developing embedded automotive content in 3D is more efficient than using 2D to mimic 3D visual effects.

For Support or Questions:
Email: cgiordano@disti.com
Phone: +1-407-206-3390 ext 129

INTRODUCTION

In the automotive industry, the economies of scale dramatically influence design decisions since savings of a few dollars per unit can quickly translate into millions of dollars saved over the production period of a vehicle. In the embedded graphics computing arena, using less texture memory in a design means the resulting embedded system hardware requires less physical memory. This paper not only describes how using 3D saves on memory, which in turn shows significant cost savings on production units, it also describes additional benefits brought about by using 3D including reduced development times and simplified asset management.

Choosing a 2D solution constrains the UI Designer into rendering a series of images that represent static positions between two desired 3D states; in effect the UI Designer renders a frame-by-frame movie to depict motion of a 3D object. In order to produce the appearance of a smoothly rotating 3D object, the UI Designer needs to produce more incremental frames for each second of motion. For completely smooth motion, the UI Designer needs to produce 60 images to represent one second of motion. This paper shows how using a 3D architecture yields significant savings on embedded memory usage when compared to using 2D practices to mimic the 3D visual effects.

A BRIEF WORD ON MEMORY

For the purposes of this paper, our discussions will focus on read-only memory, or ROM. ROM is the data storage component in computers and other electronic devices and differs from random-access memory, or RAM. RAM is referred to as volatile memory where applications run and is cleared when the power turns off.

3D SAVES MEMORY

A technique commonly used to mimic interactive 3D rendering is to pre-render the 3D content as a series of still images. Then in the target system, the pre-rendered images are played back in sequence, essentially simulating the appearance of real-time 3D rendering. This section compares the storage costs of natively rendered 3D content and simulated 3D content using pre-rendered still image sequences.

Case 1: Use 2D Rendered Textures to Represent a 3D Rotating View

- 32-bit 4-channel RGBA (Red, Green, Blue, Alpha) images: The RBG channels define the color and the transparency channel (Alpha) supports compositing.
- 512x256 display area.
- An animation created to orbit the car in 3 seconds at 60 frames per second yields 180 image files (3 x 60 = 180).

Figure 1 depicts the first 32 of 180 2D still images required to create the effect of a full 3D orbit of a car using 2D images:

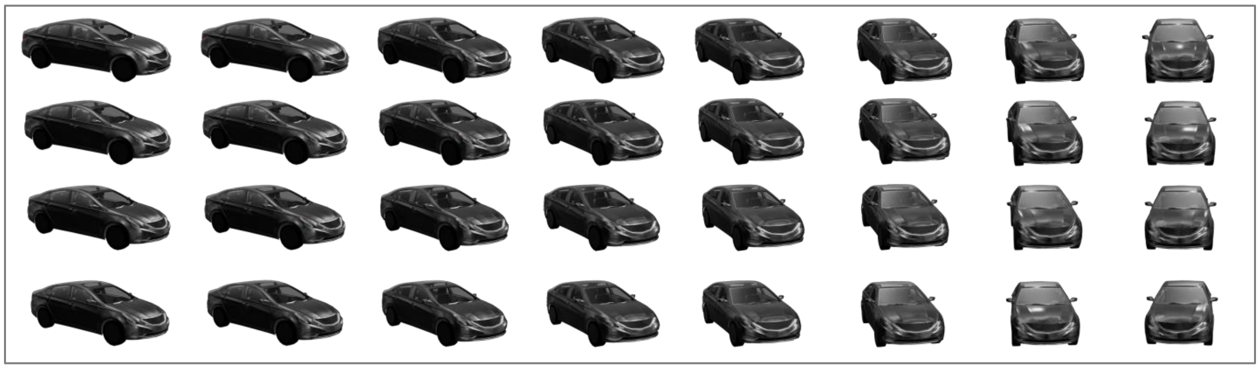


Figure 1 – Images 1-32 of 180

Shown below is the memory size calculation for the 2D image sequence. The calculations below use the per image byte size (32 bits = 4 bytes), the image height (H) and width (W), and the total number of maps required to render the rotation sequence.

$$Texture_{total} = 4\ b\ x\ H_{pixels}\ x\ W_{pixels}\ x\ Number\ of\ Maps$$
$$Texture_{total} = 4\ x\ 512\ x\ 256\ x\ 180$$
$$Texture_{total} = 94.4\ Mb$$

This yields a total **uncompressed** memory size calculation for the 2D use case at 512x256 of 94.4 Mb. Throughout this paper, texture memory calculations are measured as uncompressed since they have the lowest performance impact at runtime. While using compressed textures will save on storage memory the images need to be uncompressed at runtime in order to render. The process of uncompressing is processor intensive and negatively impacts the UI performance regardless of the development method used.

This case highlights that using the 2D method for creating a 3D representation yields a large amount of texture data; 94.4 Mb for 3 seconds of animation. This texture memory size is not optimal for running on cost effective embedded target systems given that any practical user interface application will require significantly more than 3 seconds of animation.

This memory size calculation directly correlates to the desired image size. If the UI Designer wishes to make the car larger on screen, then the 2D images will need to increase in size to maintain a 1-to-1 correlation between the image pixels and the screen display pixels (i.e., avoid over sampling). As a result, the cost per second of the animation goes up exponentially. Figure 2 shows the memory consumption for the case of the fixed 3 seconds of animation at various image sizes.

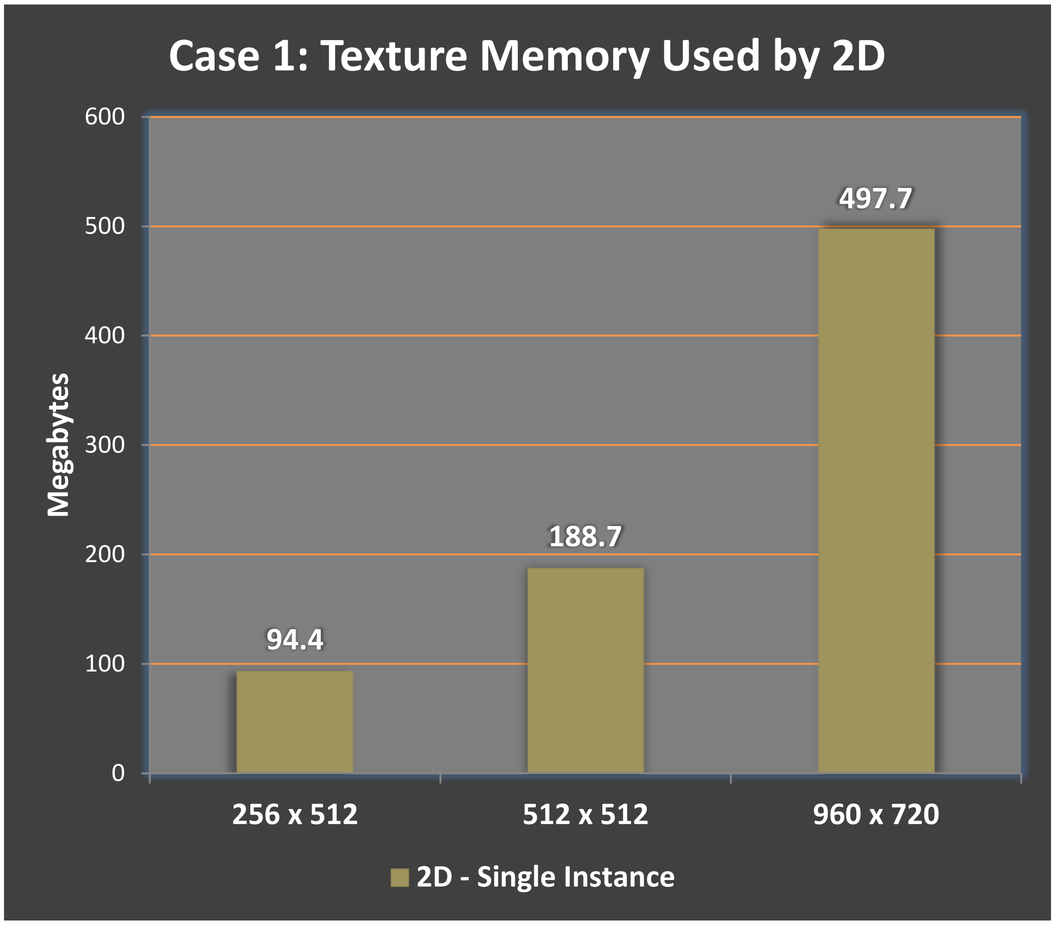


Figure 2 – Data Chart for Memory used by 2D Image Renders at Multiple Resolutions

Case 2: Use 3D Model with UV Texture Maps

- 32-bit 4-channel RGB (Red, Green, Blue, Alpha) images applied as texture maps to the model.
- 512x256 display area (is scalable without re-rendering the images)
- Created camera view to circle around the 3D object at any speed required

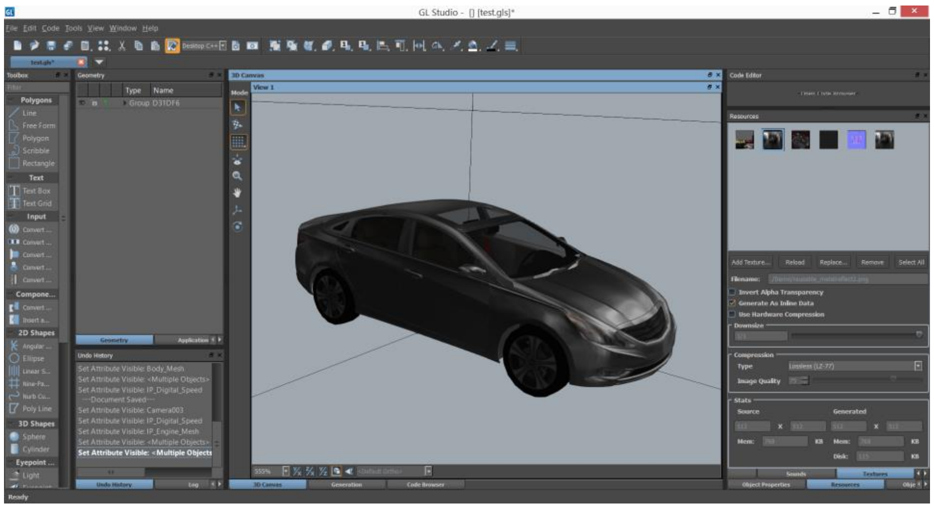
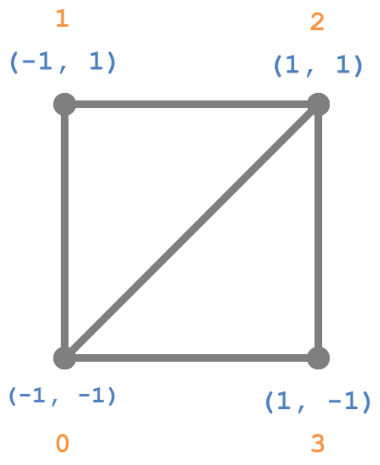


Figure 3 - View of 3D model in GL Studio used for real-time 3D rendering

The memory size of a 3D model comprises of calculating the memory required for the model's geometry and its associated texture maps. Total geometry memory is comprised of the vertex buffer size and index buffer size.

The vertex buffer is a 3D graphical container that holds the point data (position, shading vectors, colors, etc.) representing the model of the car. The index buffer is the graphical container used to "connect the dots" of each vertex to form a polygon. Figure 4 depicts an example vertex buffer and index buffer for a single square. This example assumes the Z-coordinate of the vertices is 0.



Vertex Buffer	Index Buffer
(-1, -1)	0
(-1, 1)	1
(1, 1)	2
(1, -1)	3
(-1, -1)	0
(1, 1)	2

Figure 4 – Example of a Vertex Buffer and Index Buffer

The size of each vertex is comprised of its 3D spatial coordinate (XYZ) as a 32-bit floating point number, the 2D coordinates of the mapped texture image (UV) as an unsigned short, and the directionality of the vertex normal (NNN) as an unsigned character.

$$V_{size} = XYZ + UV + NNN$$
$$V_{size} = 12_{FLT32} + 4_{USHORT} + 4_{UCHAR}$$
$$V_{size} = 20\ b$$

This yields 20 bytes for each vertex in the model. The index size is either 2 or 4 bytes depending on the number of indices in the model.

$$I_{size} = 2 \; or \; 4 \; b$$
$$If \; \langle Number \; I < 65000 \mid I_{size} = 2 \mid Else \; I_{size} = 4 \rangle$$

This example uses a worst case assumption of 4 bytes for each index in the model. The calculations below derive the total memory used by the 3D geometry of the model.

$$\begin{aligned}&Assume \; 3D \; model \; with \; 70{,}000 \; Triangles\\&Assume \; 3 \; Verticies \; per \; Triangle\\&Assume \; 3 \; Indicies \; per \; Triangle\end{aligned}$$

$$\begin{aligned}V_{buffer} &= Triangle_{\,number} \; x \; V_{size} \; x \; Verticies\\V_{buffer} &= 70{,}000 \; x \; 20 \; x \; 3\\V_{buffer} &= 4.2 \; Mb\end{aligned}$$

$$\begin{aligned}I_{buffer} &= Triangle_{\,number} \; x \; I_{size} \; x \; Indicies\\I_{buffer} &= 70{,}000 \; x \; 4 \; x \; 3\\I_{buffer} &= 0.28 \; Mb\end{aligned}$$

$$\begin{aligned}Geometry_{total} &= V_{buffer} + \; I_{buffer}\\Geometry_{total} &= 4.2 \; Mb + 0.28 \; Mb\\Geometry_{total} &= 4.5 \; Mb\end{aligned}$$

Calculations for the texture memory size used by the model are identical to the methods used for the 2D image size calculation. The calculations below assume image map sizes identical to those used in the 2D image case (512x256) and that the model will have three maps applied; one for each rendering channel, normal, diffuse, and specular.

$$\begin{aligned}Texture_{total} &= 4 \; b \; x \; H_{pixels} \; x \; W_{pixels} \; x \; Number \; of \; Maps\\Texture_{total} &= 4 \; x \; 512 \; x \; 256 \; x \; 3\\Texture_{total} &= 1.8 \; Mb\end{aligned}$$

This yields a total memory size calculation for the 512x256 use case of 6.1 Mb.

$$\begin{aligned}Memory \; Total_{3D} &= Geometry_{total} + Texture_{total}\\Memory \; Total_{3D} &= 4.5 \; Mb + 1.8 \; Mb\\Memory \; Total_{3D} &= 6.1 \; Mb\end{aligned}$$

In conducting the analysis on these two cases, using 2D textures to represent 3D content requires approximately 31.5 Mb of storage per second of animation, or in this case, 94.4Mb for 3 seconds. Using a single 3D model in the runtime scene yields 6.1 Mb of memory (1.8 Mb of which is texture memory) with no per-second animation storage cost.

Figure 5 below compares the 3D memory requirements determined in this case with the 2D memory requirements calculated in the previous case. In order to compare the two values the vertical axis in the chart changed to a **logarithmic scale** so the 3D data values are visible. This is indicative to the order of magnitude difference in memory utilization between these two cases.

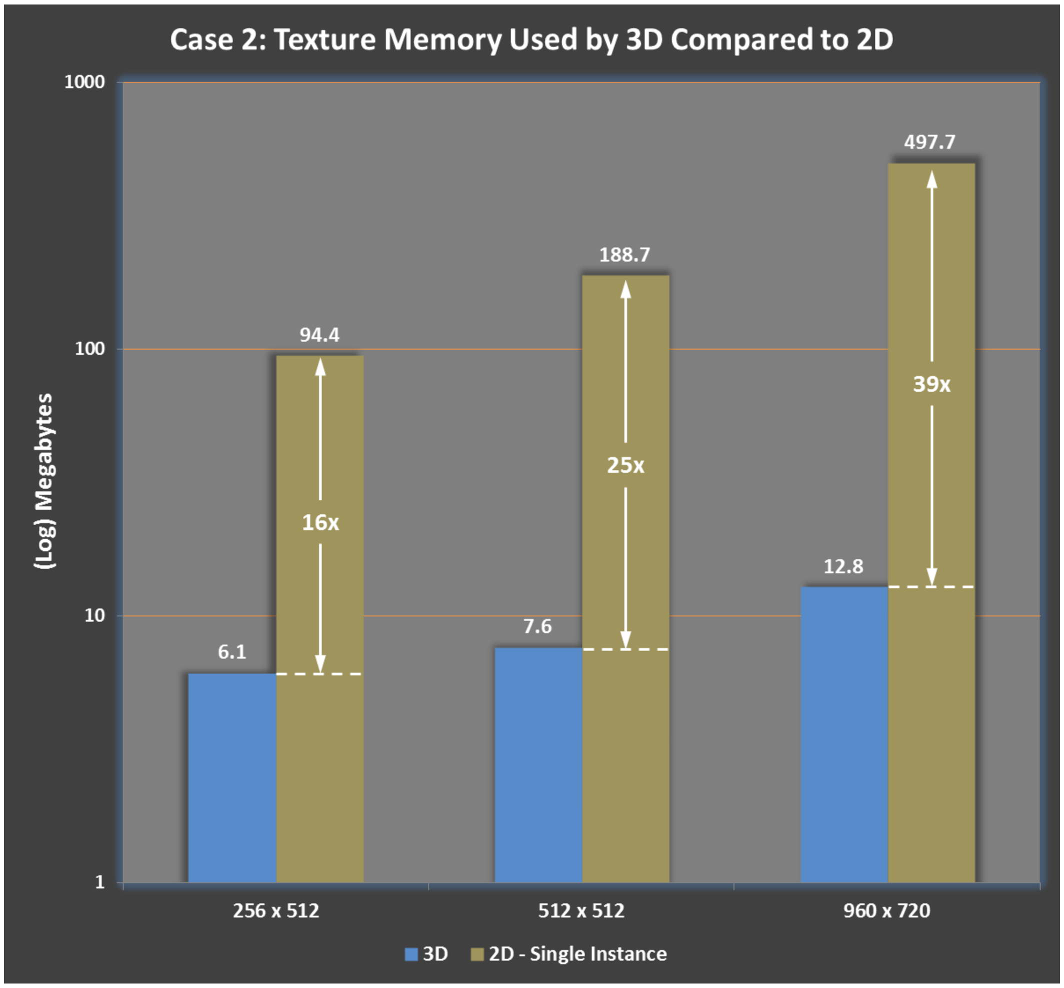


Figure 5 - Data Chart Comparing Memory used by 2D Image Renders and 3D Model at Multiple Resolutions

THE IMPACT OF ADDING DYNAMIC CONTENT

Extrapolating on the above example, if plans require adding new animations to a 2D rendered set of textures that represent a 3D view, the UI Designer must render a new set of still images for each new animation. In the 3D workflow, the HMI designer merely needs to export a set of animation key frames.

Figure 6 depicts the 2D process for adding new animations. Notice that each time the UI Designer creates a new animation it results in a corresponding set of new images, further adding to the texture memory usage and development time. This also adds to the complexity of maintaining an unnecessarily large set of assets.

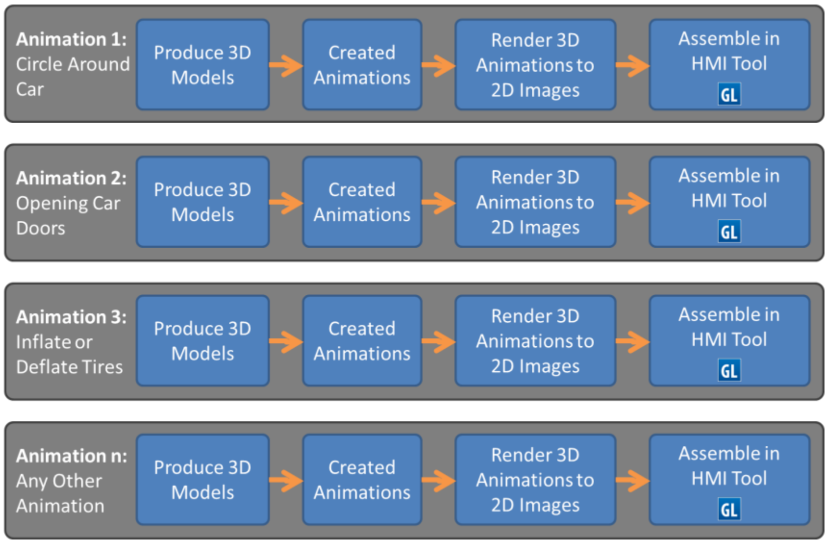

Animation 1: Circle Around Car
Produce 3D Models
Created Animations
Render 3D Animations to 2D Images
Assemble in HMI Tool
GL
Animation 2: Opening Car Doors
Animation 3: Inflate or Deflate Tires
Animation n: Any Other Animation

Figure 6 - 2D Process to add new animations

Contrasting this, Figure 7 depicts the 3D development process. In this case, the UI Designer adds new capability to the 3D object by simply adding the capability in the 3D model and exporting the new 3D behaviors via an updated key frame set for the runtime engine. The UI Designer reuses the same asset for multiple capabilities without increasing the memory load or needing to manage additional libraries of assets.

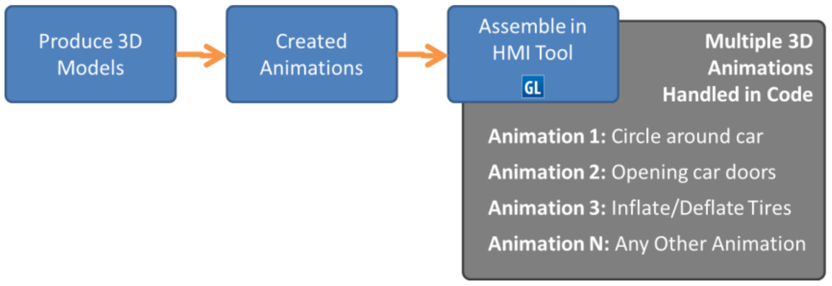

Produce 3D Models
Created Animations
Assemble in HMI Tool
GL
Multiple 3D Animations Handled in Code
Animation 1: Circle around car
Animation 2: Opening car doors
Animation 3: Inflate/Deflate Tires
Animation N: Any Other Animation

Figure 7 - 3D Process to add new animations

By deferring the rendering until execution the UI Designer no longer needs to update and maintain a library of intermediate assets. Table 1 summarizes the comparison factors between choosing to use 2D or 3D to create dynamic content.

Table 1 – Factor Comparison between 2D and 3D Dynamic Content

Factor	3D	2D
Minimal Change Modification Impact	✓	✗
Negligible Memory Impact	✓	✗
Simplified Asset Management	✓	✗
Simplified Development Process	✓	✗

DYNAMIC CONTENT REQUIREMENTS

Modern automotive applications increasingly have requirements for dynamic content. Table 2 shows examples of the kinds of properties a consumer would expect to see depicted on the car at runtime.

Table 2 – Car Properties to Control at Runtime

Property	Value
Paint Color	RGB
Head Lights	On/Off
Daytime Running Lights	On/Off
Tail Lights	On/Off
Brake Lights	On/Off
Turn Signals	On/Off
Doors	Open/Close
Hood	Open/Close
Trunk	Open/Close
Tire Pressure	RGB
Seatbelt Status	RGB

When using 2D methods with pre-rendered video, the UI Designer must render and store variation separately. Each variation multiplies the memory requirement according to the number of values the variation needs to represent. For instance, having separate versions of the model with the lights on and off doubles the storage for each set of lights, if you have 4 paint colors, it is 4 times the storage. The combinations quickly become a detriment to system memory and add an order of magnitude to the asset management. Figure 8 depicts 32 of the 180 images, shown previously, updated to include four different color schemes.

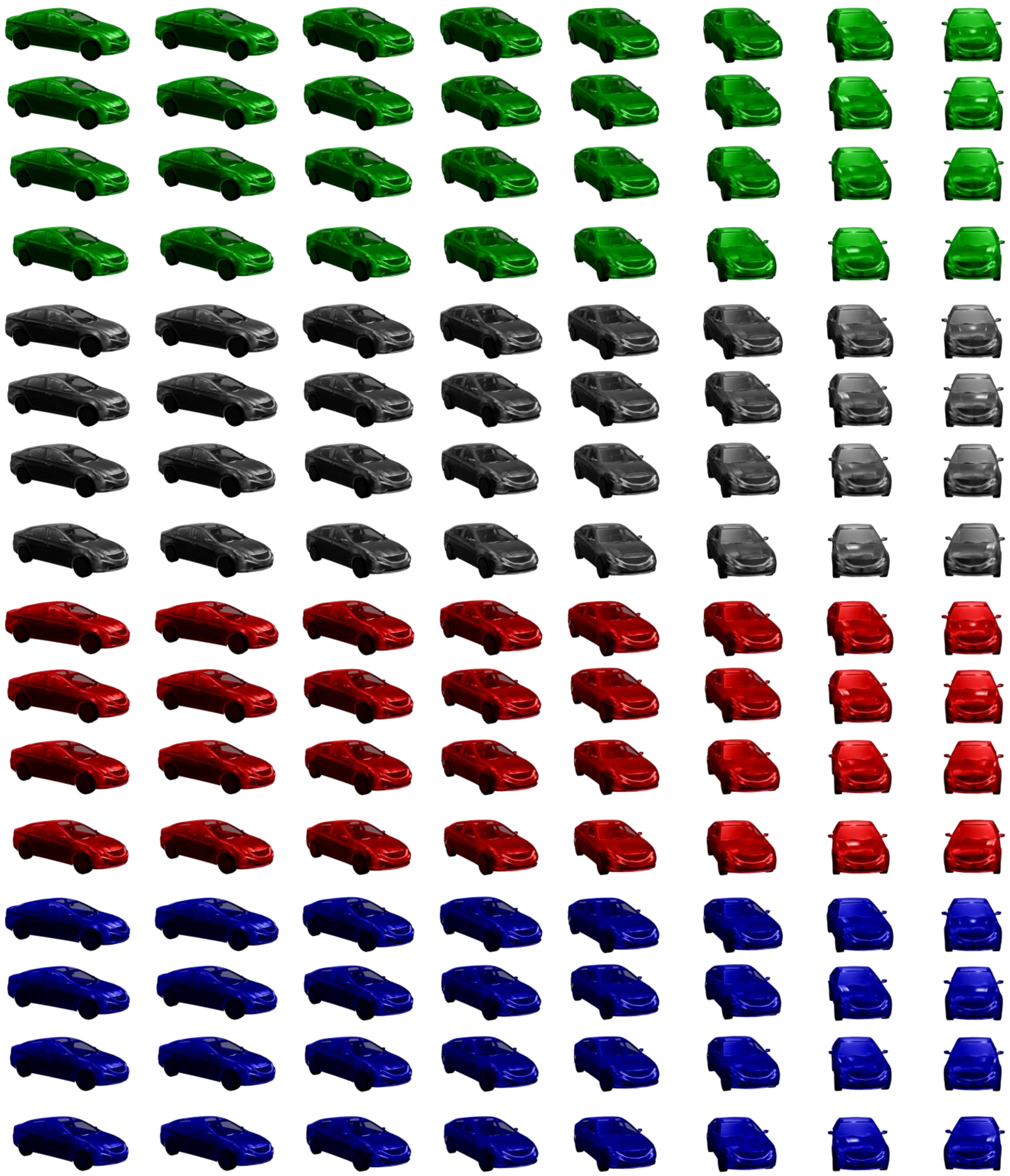


Figure 8 – Images 1-32 of 180 of Four Color Schemes

When rendering 3D models directly, the application combines and controls the various aspects of the animation without an impact on the storage requirements.

Figure 9 shows the memory impact of depicting 10 different properties of dynamic content in a 2D solution versus a 3D solution. As discussed with Figure 5, in order to compare the data values the vertical axis in the chart depicts a logarithmic scale so the 3D data values are visible.

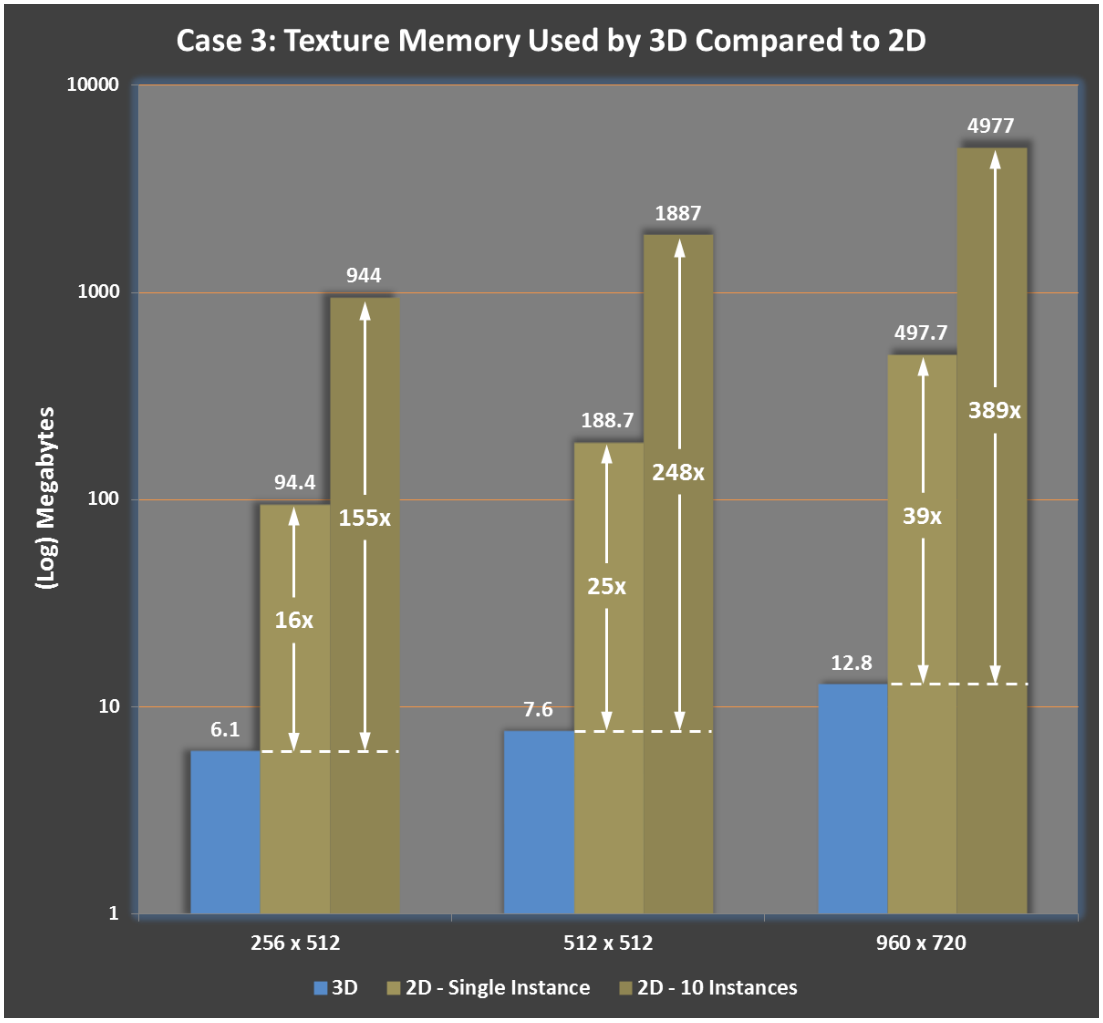


Figure 9 – Data Chart Comparing Memory used by 2D Image Renders and 3D Model at Multiple Resolutions and Multiple Instances

BILL OF MATERIAL IMPACT

Another way to look at the data depicted in Figure 9 is the impact these cases have on the deployed systems bill of material (BOM). Table 3 shows how the memory implication translates into reality for the physical memory purchase. For instance, while the 2D – 10 Instances memory demand is 4.98 Gb the BOM purchase for memory needs to be 8 Gb since memory is only available in 2^n increments.

Table 3 – BOM Memory Requirement

	Resolution		
Use Case	**256 x 512**	**512 x 512**	**960 x 720**
3D	8 Mb	8 Mb	16 Mb
2D – Single Instance	128 Mb	256 Mb	512 Mb
2D – 10 Instances	1 Gb	2 Gb	8 Gb

At an estimated $1 per Gb of memory, this translates to an extra $3 per unit for the material cost just to round off the memory allocation. Compared to the 3D high definition resolution memory requirement of 16 Mb the extra cost per unit is greater than $7. Over a production run of 2 million units, the 3D use case yields, at a minimum, a **BOM cost savings of $14 million**.

CONCLUSION

This paper described how using 3D architecture saves money by reducing system memory requirements on the deployed system, simplifying the development process, and reducing the number of project assets to manage. The design decision to utilize 3D easily translates these unit savings into substantial deployment savings once the automotive industries economy of scale occur across the entire project.