https://dx.doi.org/10.22161/ijaers.4.12.12 *ISSN: 2349-6495(P) | 2456-1908(O)*

Updating and Rendering Content on Objects in a three Dimensional Virtual Environment

Le Anh, Nguyen Trung Kien

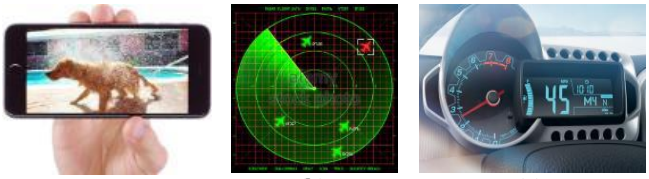
Institute of Simulation Technology, Le Quy Don Technical University, Hanoi, Vietnam

Abstract*—The virtual reality applications are increasingly popular in social life from culture, education, health to entertainment. In a three-dimensional virtual environment, the features of objects such as shape, color, movement, etc. are digitized and simulated using basic computer graphics techniques. In addition to the above characteristics, objects can contain content which changes in the form of images or text, with diverse information. The problem is that these content should be rendered on the virtual object, so that both the content and the real time of the application can be satisfied. In this paper we present an approach for updating and rendering content on objects in a three-dimensional virtual environment based on GPU architecture.*

***Keywords*— Virtual reality, texture, GPU programming.**

I. INTRODUCTION

In everyday activities, people often communicate and manipulate objects based on the content of the images and text displayed on them (*Fig. 1*). Building the objects in a three-dimensional virtual environment usually goes through two steps . Step one is object modeling that is supported by the software such as 3DSMax, Blender, etc. The second step is to render the characteristics of color and content on the objects by texture mapping [1, 2, 3]. The problem is that when the objects are rendered in the virtual environment, the change of content on the objects must also be simulated. In this paper, an GPU based approach will be presented for updating and rendering the content on the three-dimensional objects.



(a) *(b)* *(c)*

Fig. 1. Objects with the content of the images and text (a) smartphone screen, (b) radar screen, (c) speedometer

II. THE GRAPHICS PIPELINE

Texture mapping technique is used to render texture on the objects. The three-dimensional object model and texture files of the model are stored on a computer hard drive, then are uploaded to main memory by the program on the CPU. The CPU handles memory usage, controlling the flow of data between the CPU and the GPU (*Fig. 2*). Data exchange between CPU and GPU via PCI-Express (PCIe) [9, 10]. At present, PCIe 2.0 and PCIe 3.0 are two commonly used, two-way transfer rates (CPU-GPU, GPU-CPU) respectively 16GB/s and 32 GB/s. This data rate satisfies the data transfer needs between CPU and GPU.

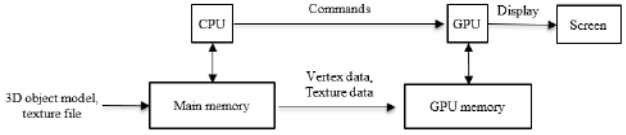


Fig. 2. Rendering texture on a three-dimensional object

Fig. 3.

Commands required to perform computation on the GPU are passed from the CPU via the command buffer. When the program on the CPU calls object drawing commands to execute on the GPU, vertex data and texture data are transferred to GPU memory, and the graphics pipeline on the GPU is executed (*Fig. 3*). On GPU memory, vertex data is stored in vertex buffers and index buffers, and texture data is stored on texture buffer. These buffers are inputs to the graphics processing steps on the GPU.

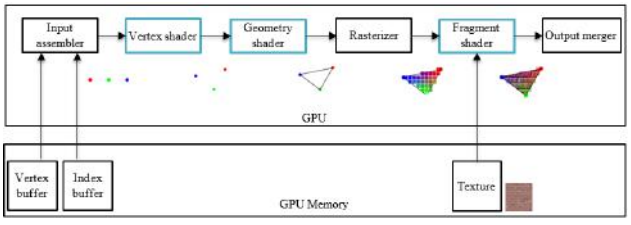


Fig. 4. The graphics pipeline

III. THE PROPOSED METHOD

Render target is a buffer that pixel processing can be executed, and the output can be used as a texture that is an input of the next step. In this section, render target is used to solve the problem of updating and rendering content on a three-dimensional virtual object. Using memory buffers on GPU memory and leveraging parallel GPU processing capabilities reduce CPU computations. At the same time, the image rendering speed is enhanced and the real-time processing is remained. We classify the contents displayed on an object into three types as follow icons, movie frames, and text.

3.1. Rendering icons

https://dx.doi.org/10.22161/ijaers.4.12.12 *ISSN: 2349-6495(P) | 2456-1908(O)*

Using icons to represent the objects on the screen or on a three-dimensional virtual object is a common choice. The information of the object such as position, direction, velocity, etc., should be updated accurately and continuously on the screen or on the surface. To do that, the position the objects must be converted from world space to screen space (*Fig. 4*).

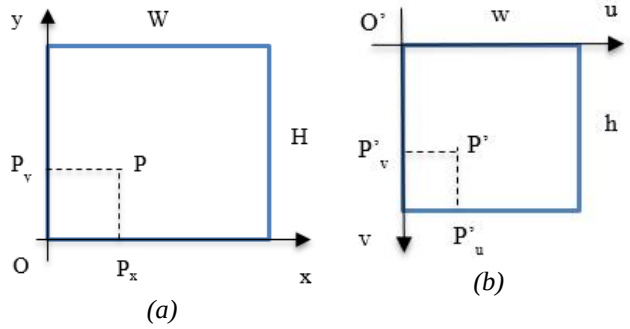


Fig. 5. P in world space (a), P' in screen space (b)

where P(P_x, P_y) is in world space, P'(P'$_u$, P'$_v$) is in screen space, W and H are respectively the width and the length in world space, w and h are respectively the width and the height in screen space. The transformation is written in the form of a matrix as follows:

$$\begin{pmatrix} P'_u \\ P'_v \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{w}{W} & 0 & 0 \\ 0 & \frac{-h}{H} & h \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} \qquad (1)$$

Here are the steps in the proposed method to render icons on the screen or on a three-dimensional virtual object:

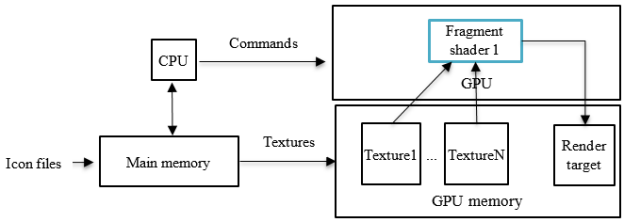


Fig. 6. Rendering icons on a render target

First, the icons are uploaded to main memory by the program on the CPU. At the same time, a render target is initialized on the GPU memory space. The size of render target is usually the power of two for optimal graphical process, especially for texture-related process.

Second, the icons would be rendered on the render target. Assume that there are N icons that are stored on main memory in the form of N textures. When the program on the CPU calls the drawing commands, the icon textures and object information such as coordinate, direction are transferred from main memory to GPU memory. On GPU memory, N textures (Texture1,..,TextureN) are the input of fragment shader 1. The fragment shader 1 performs color updates for the render target, instead of screen buffers. Specifically, icon textures are rendered on the render target based on the object information such as coordinates and direction. The result of the color update on render target is stored on the GPU memory and are reused for the next step. Note that the rotation of texture representing the direction of object can is written in the form of a matrix as follows:

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} u - 0.5 \\ v - 0.5 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \qquad (2)$$

where [u, v] and [u', v'] respectively are the texture coordinates of the points before and after the rotation.

Third, the icon textures would be displayed on the screen as shown below:

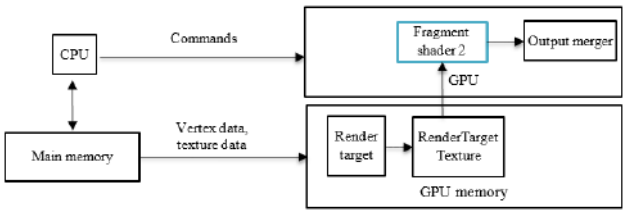


Fig. 7. Rendering RT texture on the screen or on a three-dimensional virtual object.

The data on the render target is the color value of the icon textures and is copied to another texture called render target texture (RT texture). On GPU memory, the RT texture is the input of fragment shader 2. The fragment shader 2 performs texture mapping technique to render the icon textures on the screen or on a three-dimensional virtual object. Finally, the result would be output to the screen buffer.

3.2. Rendering movie frames

The input is a movie file, the requirement is that the movie content should be displayed on a three-dimensional objects such as smartphone screen, computer screen, TV screen, etc. A movie file is a sequence of frames and each frame corresponds to a image at a given moment. Each frame has a specified width (w) and height (h). The number of frames (F) is determined by frame per second (fps) and the duration (t), ie: F = fps * t

Here are the steps in proposed rendering pipeline to render the frames on the screen or on a three-dimensional virtual object:

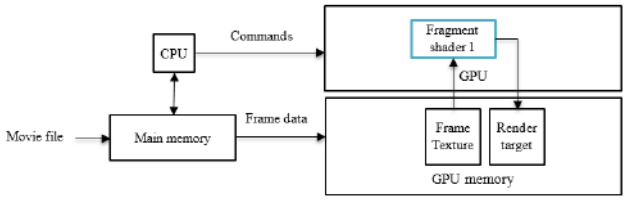


Fig. 8. Rendering frames on a render target

First, the movie file is uploaded to main memory by the program on the CPU. The properties of the video include the frame rate in fps, the size (w, h) of each frame defined in the program. At the same time, a render target is initialized on the GPU memory space, which is used to

https://dx.doi.org/10.22161/ijaers.4.12.12 *ISSN: 2349-6495(P) | 2456-1908(O)*

store the frame data. The size of render target is initialized by the size of a frame (w, h).

Second, each frame would be read by the program at each loop of rendering pipeline. When the program on the CPU calls the drawing commands, the frame data is transferred from main memory to GPU memory. On GPU memory, there is a texture containing the frame data received from the CPU, known as the Frame Texture, which is the input of fragment shader 1. The fragment shader 1 performs color updates for the render target. The result of the color update on render target is stored on the GPU memory and are reused for the next step.

The next step is the same as the final step in the previous section, except that the RT texture contains the content of the frame data stored in the render target.

3.3. Rendering text

In this paper, we use a bitmapped font to render text on the screen or on the surface of the three-dimensional virtual object because the advantages of a bitmap font are fast, flexible and platform independent. In fact, a bitmapped font is simply an atlas texture containing a collection of glyphs and symbols as follows:

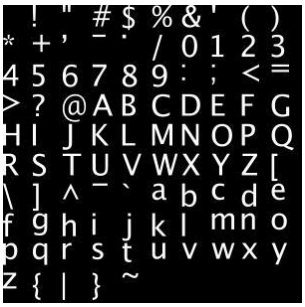


Fig. 9. Atlas texture font bitmap

The process of rendering lines of text on the screen or on a three-dimensional virtual object we perform is similar to the process of rendering icons because these character lines are essentially sets of character images that are arranged sequentially.

IV. EXPERIMENT

The computer configuration used to experiment as follows: : CPU Intel Core i5-4210H CPU 2.90GHz, RAM 8GB, graphics card Nvidia GTX850M, PCI-Express 2.0, display support H.264, VC1, MPEG2 1080p video decoder. The simulator is based on the open source engine Unreal 4.15. Applying the proposed rendering pipeline, we built the three-dimensional objects with the content that change over time. In the first implementation, *Fig. 9* shows the cockpit of UH-60 helicopter being seen from the pilot's position. The radar screen in the cockpit can scan and locate the other helicopters, where two other helicopters rendered on the screen with two blue dots (*Fig. 9*).

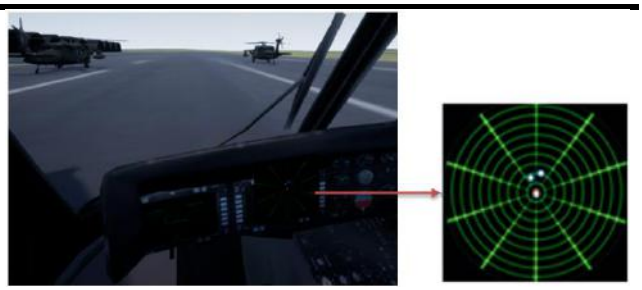


Fig. 10. Two helicopters are rendered on the radar screen by two blue dots

In the second implementation, we experimented with a movie on the TV screen in a virtual three-dimensional environment (*Fig. 10*). The video is selected in *.mp4 format, 13 seconds in length, frame rate 25 frames per second, and frame size is 1280 x 720.

Fig. 11. Rendering a movie on the TV screen

In the third implementation, the chosen objects with contents in the form of characters and numbers are a digital clock (*Fig. 11.a*) and an electronic LED panel (*Fig. 11.b*). Time clocks represent the types of clocks that have digital displays such as temperature meter, moisture meter, pressure meter, speedometer, etc. For electronic LED panels, the content is text shown on the LED panel. These lines can be run from left to right, top to bottom or vice versa, by changing the texture coordinates of the pixel in the fragment shader:

$$(u', v') = (u, v) + (uOffset, vOffset) \quad (3)$$

where uOffset and vOffset are the time intervals.

(a) *(b)*

Fig. 12. Rendering numbers on a digital clock (a), rendering text on an electronic LED panel

V. CONCLUSION

Taking advantage of the support for programming on graphics cards, we proposed an approach to update and render content on a three-dimensional virtual objects based

https://dx.doi.org/10.22161/ijaers.4.12.12 *ISSN: 2349-6495(P) | 2456-1908(O)*

on GPU programming. The change of content is updated to a render target, which is a memory location located on the video memory. The result of the render target is then used as an input of texture mapping on a three-dimensional virtual object. Most of the computations are done on the GPU so the computations on the CPU have been reduced to improve the rendering speed and remain real-time processing.

REFERENCES

[1] Gerald Farin , State of the art in 3D modeling. In Proceedings - 5th International Conference on Frontier of Computer Science and Technology, FCST 2010 [5577356] DOI: 10.1109/FCST.2010.114, 2010.

[2] Georgios Kordelas , Juan Diego P`erez-Moneo Agapito, Jes`us M. Vegas Hernandez , and Petros Daras, State-of-the-art Algorithms for Complete 3D Model Reconstruction, University of Valladolid, Valladolid, Spain.

[3] Catmull, E. A subdivision algorithm for computer display of curved surfaces, University of Utah, 1974.

[4] Wojciech Matusik, Frédo Durand, Texture Mapping & Shaders - Computer Graphics Course, MIT OpenCourseWare, http://ocw.mit.edu, Fall 2012.

[5] Magnenat-Thalmann, Nadia, Thalmann, Daniel (Eds.), State-of-the-art in Computer Animation, Proceedings of Computer Animation '89, Springer Japan, eBook ISBN: 978-4-431-68293-6, DOI: 10.1007/978-4-431-68293-6, 1989.

[6] Nicolas P.Rougier (INRIA), Higher Quality 2D Text Rendering, Journal of Computer Graphics Techniques, vol.2, No.1, 2013.

[7] Charalampos Koniaris, Darren Cosker, Xiaosong Yang, Survey of Texture Mapping Techniques for Representing and Rendering Volumetric Mesostructure, Journal of Computer Graphics Techniques, vol.3, No.2, 2014.

[8] F.Klose, O.Wang, J.-C.Bazin, M.Magnor, A.Sorkine-Hornung, Efficient GPU Based Samping for Scene-Space Video Processing; Vision, Modeling, and Visualization, 2015.

[9] John Nickolls, David Kirk (NVIDIA), Graphics and Computing GPUs.

[10] John Nickolls, William J.Dally (NVIDIA), The GPU Computing Era, IEEE Micro (Volume: 30, Issue: 2, March-April 2010, ISSN**:** 0272-1732, DOI: 10.1109/MM.2010.41.