MIX RENDERINGS TO FOCUS PLAYER'S ATTENTION

Vincent Boyer, Jordane Suarez and Kamel Haddad
L.I.A.S.D.
Université Paris 8,
2 rue de la libérté
93526 Saint-Denis, France
E-mail: boyer@ai.univ-paris8.fr
[dex, pepitox]@bocal.cs.univ-paris8.fr

KEYWORDS
Rendering, GPU programming, Non-Photorealistic Computer Graphics.

ABSTRACT

We propose to mix different rendering techniques in a same frame in a video game to focus player's attention. Mixing different kinds of rendering techniques in the same frame permits to enhance informations on 2D images or 3D scenes. Images in video games are essential and are the main vector of informations for the player. With our model, in high-scale scene, some informations can be enhanced. Our model allows the designer to choose the rendering techniques used simultaneously; to build the GPU program that mix theses renderings and export it. The generated GPU program achieves the rendering of the scene with rendering techniques selected by the designer in one pass. This new GPU program can be imported by the designer in the video game and can be used while key points are given. Other values, like associated distances to each rendering technique used can be changed dynamically in the video games without any changes in the GPU program. Examples are given and the images produced by our system are discussed.

INTRODUCTION

Images are the main informations vector in video games. Generally, one and only one rendering is used in a video game. For example celshading (Lake 2000) is used in GTA, and photorealistic rendering with High dynamic range imaging and Tone mapping (Mantiuk 2004) are used in other car video games. One of the main ideas of the paper is to consider that we can use simultaneously more than one rendering technique at each frame to render the scene in video games. Many rendering techniques can be used simultaneously: artistic shading like Hatching (Winkenbach 1996), (Hertzmann 2000); deformation of the 3D mesh according to the viewpoint (Rademacher 1999); bump mapping, wood texturing (Rost 2004), classical gouraud shading, texturing...

As one can imagine, this can be used to focus player's attention on a path or an objective. Or it can be used to create more complex level on a video games.

In computer graphics this idea is recent but in architecture for example, this technique is commonly used to create architectural models in city planning: new buildings are designed with details (colors, windows, trees, flowers) while other existing constructions are designed with their shapes only.

We have proposed a method to mix different rendering techniques (Boyer 2007) for 3D scene visualization. The rendering techniques used are imported by the designer and a GPU program is automatically and dynamically generated to mix selected rendering techniques. Based on GPU programing, this model is real time for large scale scenes visualization.

In the following, we present the general algorithm of our model with the automatic generation of GPU programs. Then we present the method used to optimize and export GPU programs. Examples on video games are then detailed.

OUR MODEL

In this section we describe briefly the model used to mix different rendering techniques (Boyer 2006). The description is given to introduce terms used in the following.

In this model, the user specifies:

1. *key points* which are points of the 3D space. In the following *n* is the number of key points given by the user;
2. *used renderings* and for each of them four distances *d0*, *d1*, *d2* and *d3* where [*d0*, *d3*] is the range within the rendering is used :
    - *d0*: the minimal distance in order to use this rendering;
    - *d1*: the minimal distance where we maximize the use of this rendering. Between *d0* and *d1* the rendering will be shaded;
    - *d2*: the maximal distance where we maximize the use of this rendering. Between *d1* and *d2* the rendering will be maximized;
    - *d3*: the maximal distance in order to use this rendering. Between *d2* and *d3* the rendering will be shaded.
3. a *mode* to consider key points:
    - *points*: key points are considered independently;
    - *polyline*: the key points form a polyline;
    - *polyline loop*: the key points form a closed polyline.

The GPU program that blends renderings is performed both in the vertex and the fragment shaders. The vertex shader is used to pre-compute values needed in the fragment shader where the main process is realized. The general form of the algorithm is:

```
For a given fragment
    1. Compute the closest distance d to the key points according to the mode used.
    2. For each rendering
        ④ Compute v, the weighted value to be applied to the rendering depending on the distances given by the user and d.
        ④ Compute the color depending on the rendering chosen by the user and weight it by v.
    3. Sum the colors previously obtained and clamp it between 0 and 1 for each component.
```

This model has been extended to

generate automatically and dynamically GPU programs (Boyer 2007). Softwares, like ShaderGen, Rendermonkey, ShaderDesigner allow the user to write shaders (GLSL) through a convenient GUI. A module was created to import directly GPU programs created with these softwares.

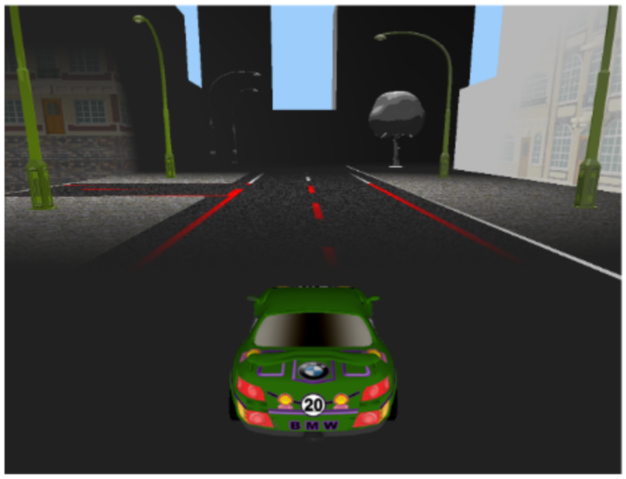
This process is realized using a parser and a generator. At the end of this process, a new vertex shader and a new fragment shader are created, compiled and linked to form the new GPU program. It is able to mix in one pass rendering techniques available and the new one imported by the designer. The new GPU program is then dynamically sent to the graphic card and replaces the existing. This process can be repeated while GPU capabilities are available.

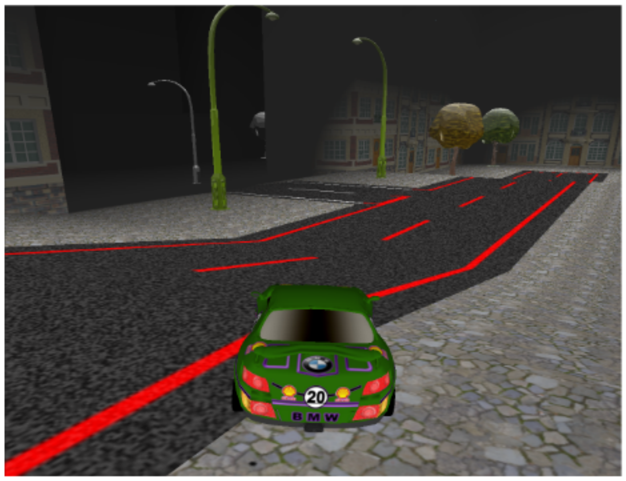
Generation and export GPU program
The GPU program created by the designer is designed to be import in a video game to focus player's attention. At the previous steps, the designer has imported and selected these rendering techniques to be mixed and chooses a mode and the different distances according to the model of the scene. Then we propose to automatically export this shader.
A new shader is then automatically created where only rendering techniques selected by the designer are integrated. This supposes that according to the mode and the number of key points given by the designer only one kind of distance computation is achieved (part 1 of the algorithm). Then we copy the functions needed (i.e. each of them corresponds to one rendering technique) and we change the number of iterations in the loop of the algorithm previously presented (part 2 of the algorithm). Only parameters (uniform, attribute) needed by the GPU program are preserved. Finally the vertex and fragment shaders are created and a description file, written in **XML** is generated to save the distances associated to each rendering, the mode used and the parameters (uniform, attribute, texture).

Then, in the video game the GPU and its description file are imported. The vertex and fragment shaders are compiled, linked and the GPU program is used to render the scene. Moreover as the parameters are specified in the file description, they can be modified dynamically to change the rendering effect. Remark that the number of key points used can be changed according to the specifications of the designer but their positions can be modified dynamically.

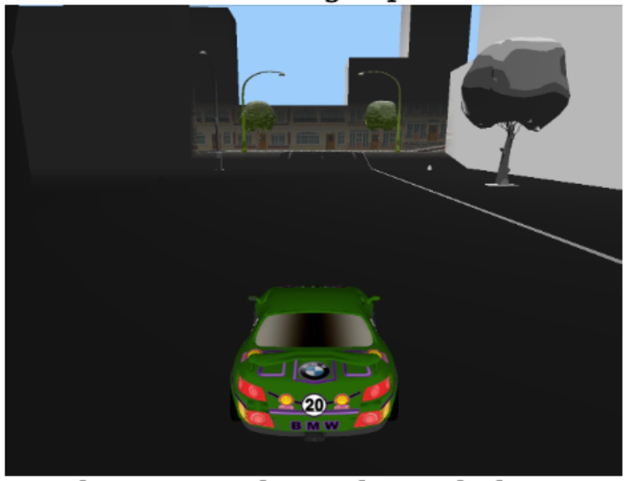
IMAGES AND RESULTS

In this section we present some images produced by our model. Those have been produced on a PC pentium IV 3.6Ghz with 1Go of memory and a nvidia graphic card Quadro FX 1400.
This snapshot comes from a car video game. The user should follow a path in a street. The path is represented by red lines along the street. In this example we have mix texture rendering and celshading. The key points are placed at the junction of two streets of the path. According to the distance texture, celshading or a mix of these ones are used to render the geometry.

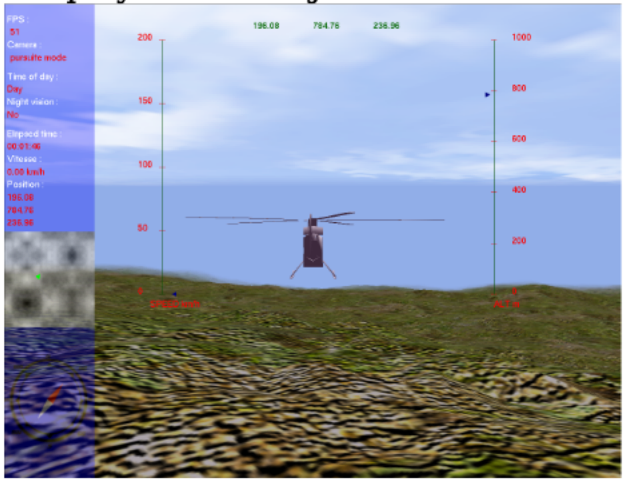




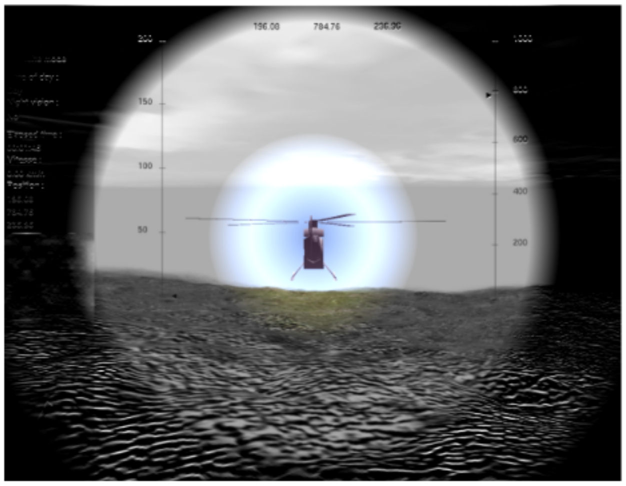
Here we present another view. As one can see on the building, renderings are mixed. Last view, the user is not on the right path !



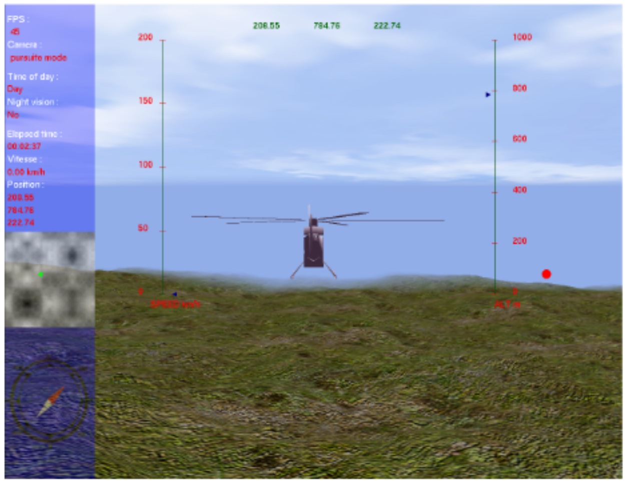
Another example with an helicopter simulation video game. Here the designer chooses to mix 2D (luminance and sobel filter) and 3D rendering techniques (textured rendering). We first present the scene rendered with one rendering technique and the same with mixed renderings. Remark that with the same GPU program during the video game execution, in the first image produced with our model one key point is used and in the second one two key points are used. The second one can be used for example to guide the player to an objective.

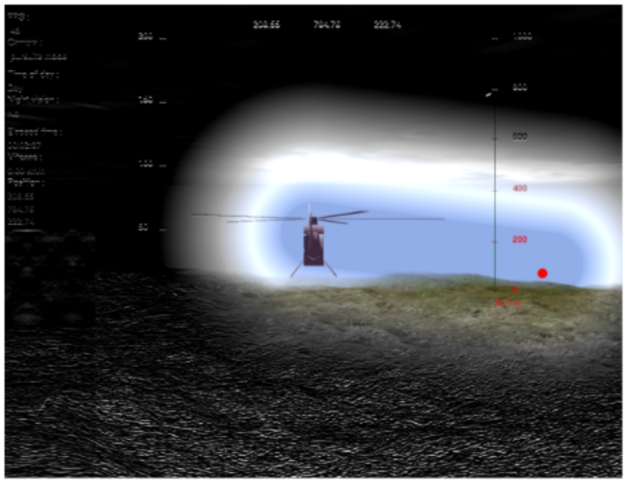


FPS :
51
Camera :
pursuite mode
Time of day :
Day
Night vision :
No
Elapsed time :
00:01:46
Vitesse :
0.00 km/h
Position :
196.08
784.76
236.96



In these cases, the frame rate decreases to 20% (car) or 10% (helicopter) . The model performs a GPU program according to GPU program selected by the designer. So generally, the performance of the generated GPU program is equivalent to the slower GPU program imported.





CONCLUSION

In this paper we have presented a rendering method to focus player's attention in video games. It is based on a model that mix different rendering techniques in the same frame. The GPU program is automatically generated and can be imported in the video game. Moreover parameters of the GPU program can be modified dynamically in the video game.

REFERENCES

Boyer V. Automatic and dynamic generation of GPU programs to mix renderings and enhance informations on high scale scenes, *In IASTED Visualization, Imaging, and Image Processing (VIIP) 2007*, Palma de Majorque, Spain.

Boyer V and Sobczyk D. Enhancing information on large scenes by mixing renderings. *In International Symposium on Visual Computing (ISVC) 2006*, South Lake Tahoe, USA.

Hertzmann A. and Zorin D. Illustrating smooth surfaces. *In Proceedings of SIGGRAPH 2000*.

Lake A, Marshall C, Harris M, and Blackstein M. Stylized rendering techniques for scalable real-time 3d animation. *In Non-Photorealistic Animation and Rendering 2000 (NPAR '00)*, Annecy, France.

Mantiuk R., Krawczyk G., Myszkowski K., Seidel H-P. Perception-motivated High Dynamic Range Video Encoding. *Proceedings. of SIGGRAPH '04 (Special issue of ACM Transactions on Graphics)*, 2004.

Rademacher P. View-dependent geometry. *In Proceedings of SIGGRAPH 1999*.

Rost R. OpenGL Shading Language. *Pearson*, 2004.

Winkenbach G. and Salesin D.H. Rendering parametric surfaces in pen and ink. *In Proceedings of SIGGRAPH 1996*.