KAPITEL 9 / *CHAPTER 9* [9]
FEATURES OF PARALLELIZING THE COMPUTING PROCESS OF FORMING THREE-DIMENSIONAL GRAPHIC SCENES

DOI: 10.30890/2709-2313.2024-32-00-044

Introduction

Rendering [1-6] of the big-scale images requires lots of computations. Even modern computers that are supplied with powerful GPUs [7-10] are not always able to handle such a high load. It is becoming an even more serious issue when it comes to mobile devices tablets and cell phones.

There are four directions on how to cope with such a problem. It is horizontal or vertical scaling and a trade-off between a client and a thin client. The most promising approach, considering limitations of the processing power of the mobile devices as well as vertical scaling limitations, considered using horizontal scaling and parallel computing, in the case of processing of graphics – parallel rendering [11-15].

9.1. Rendering parallelization approaches

There are three main approaches to parallelizing while creating realistic images: functional, spatial, and structural (Figure. 1).

Functional parallelism is supported by a pipeline architecture for executing sequential phases of geometric processing. This approach limits parallelism to the number of individual functional computational modules.

Spatial parallelism involves dividing the object space into separate volumetric elements that can be processed independently. This approach includes two main methods [16]:

[9] ***Authors:*** *Romanyuk Oleksandr Nykyforovych, Bobko Oleksii Leonidovych, Romanyuk Oksana Volodymyrivna*

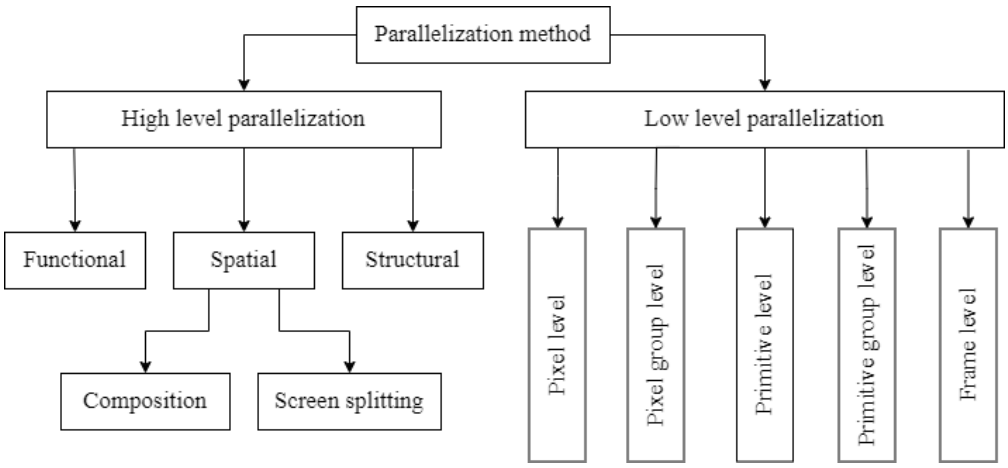

Figure 1 – Parallelization method

Image composition, where each renderer generates a full-screen image of a part of a complex scene, and the final image is obtained by overlaying the partial images.

Screen splitting, where each renderer generates a full image of a portion of the screen, and the final image is created by "stitching" different parts together.

Each method has its pros and cons. The composition scheme requires complex and expensive compositing hardware, which limits the number of possible attributes in a pixel. Additionally, the workload on the renderers may be unbalanced. On the other hand, the screen-splitting scheme assumes the presence of a redistribution grid that allows polygons for rasterization in different screen parts to be arbitrarily distributed from geometry processors. Conversely, assembling different sections into a complete image is relatively easy, as it does not require pixel composition operations.

Structural parallelism is implemented through a logical subdivision of scene elements. Each scene object, having its specifics (e.g., representation form), is marked, and each processor in the processing structure responds only to requests from objects of its class. This approach is effective, especially when simultaneously modelling natural and artificial objects, due to the use of different visualization methods.

The specific technical implementation of the rasterization subsystem is determined by basic visualization algorithms and required performance characteristics. The spatial approach to parallelization in real-time image generation has become the most widespread, where spatial parallelism is supported by a pipeline architecture.

Key levels of parallelism that are used when distributing different stages of

realistic image creation (lower-level parallelization) include:

1. Pixel level (EZ), where parallel calculation across six coordinates (x, y, z, R, G, B) and access to the frame buffer can be ensured.

2. Pixel group level, where intensities and coordinates are calculated in parallel for several pixels (this type of parallelism is implemented in ray-tracing structures).

3. Primitive level, where parallelism is implemented, for example, within a triangle, providing parallel computation for its horizontal and vertical linear or rectangular fragments.

4. Primitive group level, where parallelism is ensured by simultaneously rasterizing a set of primitives.

5. Frame level, where parallelism is achieved by simultaneously processing several subframes or complete frames.

9.2. Parallelization with the Gouraud method

In the Gouraud method [1, 15], which has become the most widely used in graphic accelerators, intensity values for polygonal vertices are calculated, which are then linearly interpolated along the edges and scan lines in the rasterization process [15]. The algorithmic and circuit-technical simplicity of the Gouraud method encourages scientists to continue work on improving this approach. A significant increase in the productivity of the Gouraud method [1] can be achieved due to the parallelization of the computational process. At the same time, it is important to achieve a balanced loading of the component dyers.

Two methods of parallelizing Gouraud rendering [15] have been developed, which use the property of constancy of the increase in color intensity along the raster lines, which determine the direction of addressing the internal points of the triangle when it is painted. The first method of painting is the technique that consists of the independent determination of color intensities in the even and odd points of the raster line between the left and right edges of the triangle, which ensures an increase in

painting productivity up to two times (Figure. 2, *a*) [15]. The second method is based on the distribution of computational actions of the coloring procedure at the lower level when the graphics processor is given the parameters of the matching triangles, as well as the color intensity at their vertices. The area bounded by the triangle is colored using Serpinski triangulation the first of order According to with method triangle breaks down into four comparable by connecting the midpoints of its sides. As a result, four triangles of equal area are obtained (Figure. 2, *b*). Comparable triangles are painted in parallel. At the same time, only one shader is used, which tints one of the triangles, and the results of his work are transferred to all other triangles with a certain shift in coordinates and corresponding transformations color intensities of points. Due to the parallelization of the computing process, the initial triangle will be painted four times faster.

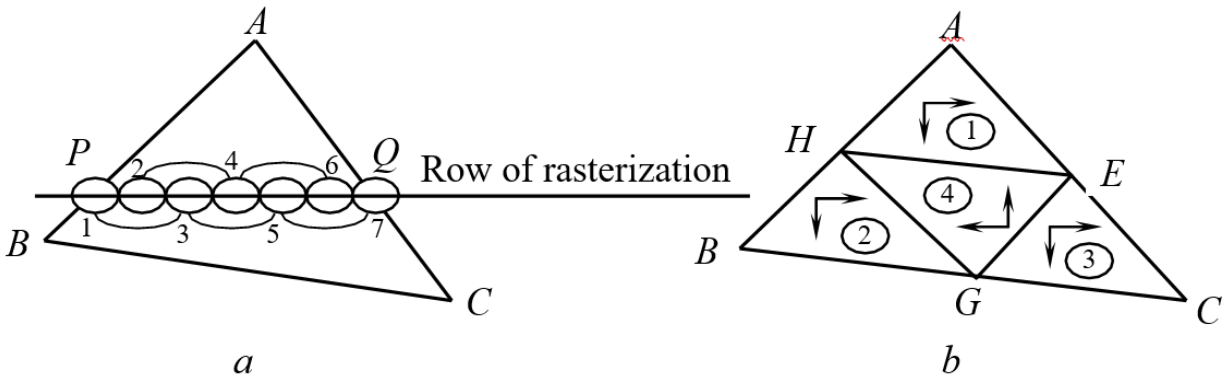

Figure 2 – Parallelization principles

Modern graphic accelerators require an even greater increase in coloring productivity, which determines the urgency of the search for new approaches and hardware solutions. Therefore, the development of structures for the parallelization of the computing process during coloring according to the Gouraud method [1], which ensures the formation of a whole segment of pixels in the raster line, acquires special importance.

The proposed approach allows for an increase in the speed of the coloring procedure due to the simultaneous determination of the color intensities of the entire

segment of points in the raster line of the triangles on which the object was previously triangulated. When implementing segmental coloring, the property is used, which consists in the constancy of the increase in the intensity of the color of the triangle along the raster line between the points of its left and right edges. This allows you to determine the intensity of each i_{th} point relative to the beginning of the segment using the formula (1)

$$I_J = I_0 + i\Delta I_g \quad (1)$$

I_0 – intensity of the first points in the segments,

ΔI_g – increased intensity along line the rasterization

The optimal size of the segment is determined, if its value even to the power of two.

The easiest way to get the value for $i\Delta I_g$ with use of multiplication blocks though this approach requires more hardware resources. It is more effective to find value for $i\Delta I_g$ with addition and subtraction operands like $2^w\Delta I$, that is very easy to find by shifting $i\Delta I_g$ towards higher bits.

Imagine p is a bit rate of the segment. Then $\log_2 p$ shifts of increase, $i\Delta I_g$ of the color intensity is done towards higher bits. Considering that operand $i\Delta I_g$ is given, the amount of $2^w\Delta I_g$ operands will be equal to k = log2p + 1, where w<k.

For example, it is possible to consider the coloring of the triangle under the condition that p = 8. It is advisable to consider this case in more detail since the bit rate of modern video memory chips has exactly this value.

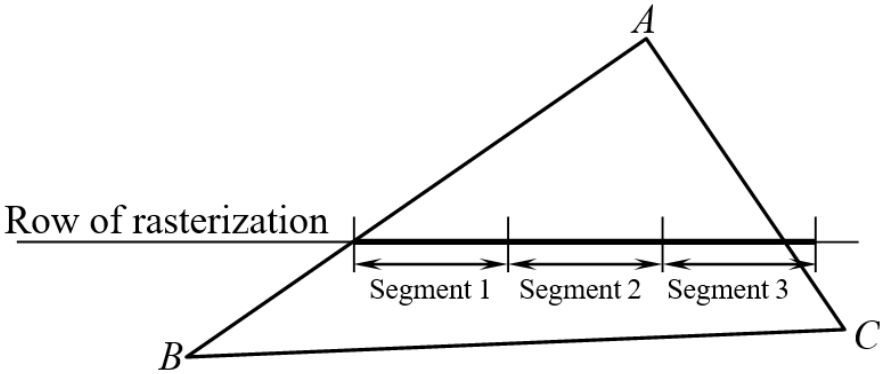

Figure 3 – Example breakdown line rasterization into segments

The beginning of the first segment coincides with the point of the left edge of the triangle on its raster line (Figure. 3). The number of segments is equal to [m/8], where m is the number of points in the raster line between the left and right edges of the triangle.

At defined intensities color points current segment expedient find and intensive grayness primary points next segment, because strategy, by which final point of one segment is the starting point of the next, will lead to the fact that the size of the segment will not be a multiple of the power of two.

To find the intensities of the points of the segment, it is necessary to determine the value of the intensity increments $2\Delta I_g$...$8\Delta I_g$.

Functional scheme block formation increments intensities (BFPI) color which includes register for storage growth intensity ΔI_g and four adders (Figure. 4). On you-during the register assembly a shift gets three value of increments $2\Delta I_g$, $4\Delta I_g$, $8\Delta I_g$. On you all the necessary values of the intensity increments are obtained during the operations of the adders and the BFPI register. It should be noted that the adder Σ_4 works in the mode of subtraction of numbers in the complementary code.

Value increments intensities served on bloc formation intensities of the colors of the segment points, where the color intensities of all points of the current segment and the color intensity of the first point of the next segment are formed (Figure. 5).

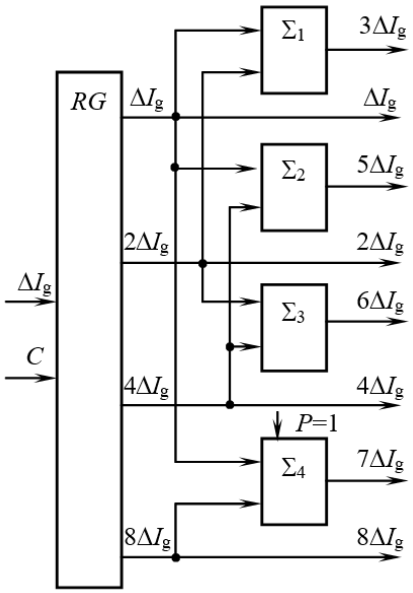

Figure 4 – Functional scheme block forming increments of color intensities

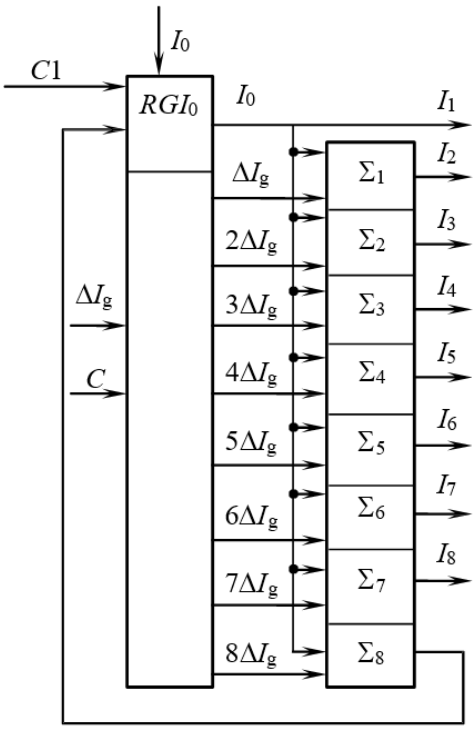

Figure 5 – Functional scheme block formation of the color intensities of the segment points

After formation current segment value color intensity with exit Σ_8 back-sits down in the register and is used for the definition color intensities of the following segment.

The length of the triangle raster line between the points of its left and right edges may not be a multiple of the length of the segment, i.e. the endpoint of the last segment may not coincide with the end point of the triangle raster. To block the recording of the points of the last segment that does not belong to the line of the triangle area, a mask register is additionally entered (the value 1 in it will indicate that the recording of the color intensity of the segment point is allowed in the video memory). The three lower digits of the binary representation *m* indicates the digit number in the mask register, from which zeros must be written to block the recording of the color intensity values of those points of the segment, which do not belong to the triangle.

The proposed structure allows you to significantly increase the performance of coloring and can be recommended for implementation in high-performance graphic systems for the formation of realistic images.

9.3. Parallelization with GPU

Parallel rendering on GPUs [7-10] allows the scene formation process to be distributed across multiple tasks. The number of cores in modern GPUs can vary significantly depending on the model and manufacturer. For example, the GeForce RTX 4000 Series has 16,384 CUDA cores, while the Radeon RX 7000 Series features 6,144 Stream processors. NVIDIA A100 series GPUs have over 6,000 CUDA cores, and the RTX A6000 series has 10,752 CUDA cores. The number of cores determines a GPU's ability to handle parallel tasks, meaning the more cores, the higher the rendering and graphics processing performance. However, it is important to note that performance also depends on other factors such as clock speed, memory size, and the architecture of the card.

GPU architecture is optimized for performing uniform calculations on large data sets (SIMD – Single Instruction, Multiple Data). GPUs can execute parallel operations on pixels, vertex data, and textures. This achieves high performance when rendering complex scenes, as simultaneous calculations can be used to render multiple objects. GPUs can also use different types of shaders (vertex, fragment, geometry) to manage the rendering process, with shaders being able to run in parallel, significantly boosting performance. Graphics processors have their own video memory, which is used to store textures, vertex buffers, shaders, and more.

Task distribution on a GPU is a crucial step for efficient parallel data processing, particularly in tasks related to graphical rendering, computational science, and artificial intelligence. A GPU has many cores that can execute thousands of threads simultaneously. A task is divided into smaller fragments that are processed in parallel. Developers can utilize these threads to distribute the computational load, which is implemented via shaders in graphics applications or specialized code blocks in computational applications. Tasks are often split into blocks, each containing several threads that perform the same operation. This approach allows more efficient use of caches and shared GPU memory since threads within a block can quickly exchange data.

Using data structures optimized for GPUs enables better management of memory distribution and data access. This may include dynamic arrays or buffers that can automatically scale and be distributed across different GPU cores. Systems can dynamically monitor and analyze the load on the GPU, distributing tasks based on this data to avoid overloading specific cores and ensure even load distribution across all resources.

Asynchronous operations allow the GPU to perform tasks without waiting for the CPU or other system components to complete their processes. This can significantly improve overall performance by minimizing idle time and increasing parallelism. Resource allocation methods involve distributing GPU resources to specific tasks or applications, ensuring optimal use of the GPU when running multiple processes simultaneously.

More advanced systems can use task graphs to determine dependencies between different computational tasks and optimally distribute them across the GPU, taking those dependencies into account.

9.4. Multiple GPUs on a single motherboard

To parallelize the images rendering, not one, but several video cards are often used. SLI is a technology developed by NVIDIA that allows two or more NVIDIA-based graphics cards to work together. This technology can use different methods of load distribution, such as Alternate Frame Rendering (AFR), where each card processes successive frames, or Split Frame Rendering (SFR), where a frame is divided into parts that are processed by different cards. CrossFire is a similar technology by AMD that allows you to connect two or more AMD-based video cards. The technology also supports various load distribution methods, including AFR and SFR, as well as Super Tiling, where the frame is divided into smaller pieces to balance the load between cards.

Comparative analysis of NVIDIA SLI and AMD CrossFire technologies

represented (Table 1).

Table 1 – Comparison of NVIDIA SLI and AMD CrossFire

Criterion	NVIDIA SLI	AMD CrossFire
Principle of operation	Uses parallel processing to improve graphics performance.	Also uses parallel processing, but with some differences in architecture.
Compatibility	Supports only NVIDIA video cards, usually requires the same model.	Supports AMD video cards, but it is possible to use different models (but with limitations).
System requirements	A motherboard with SLI support, a powerful BJ is required.	A motherboard with CrossFire support is required, as well as a powerful BJ.
Settings	Usually easier to configure via NVIDIA drivers.	May require more manual tweaking, especially for optimization.
Optimization in games	Highly optimized in most new games, often supported by developers.	Optimization varies from game to game, but some new designs also support CrossFire well.
Performance factor	Can provide up to 90% increase in performance in games.	Can provide up to 70-80% increase, depending on the game.
Performance issues	Possible problems with optimization in old games.	May be less of a problem but depends on specific GPU models.
Cost	Usually more expensive solutions due to the cost of NVIDIA video cards.	Can be a more economical choice, especially when choosing more affordable models.
Support service	Strong support from NVIDIA with frequent driver updates.	Support from AMD, but sometimes with slower updates.
Potential disadvantages	Possible problems with heat generation and energy consumption.	Possible optimization problems in some games, as well as dependence on the specifications of the motherboard.

NVIDIA SLI (Figure. 6) is better suited for users who need maximum performance in new games and high optimization. The system is easier to configure and has good support. However, the cost of solutions can be higher. AMD CrossFire (Figure. 7) can be a less expensive option that allows you to combine different models of video cards. However, the optimization may be less stable in some cases and the

cost may vary.

Figure 6 – Nvidia SLI

Source: https://www.dignited.com/58471/nvidia-sli-vs-amd-crossfire/

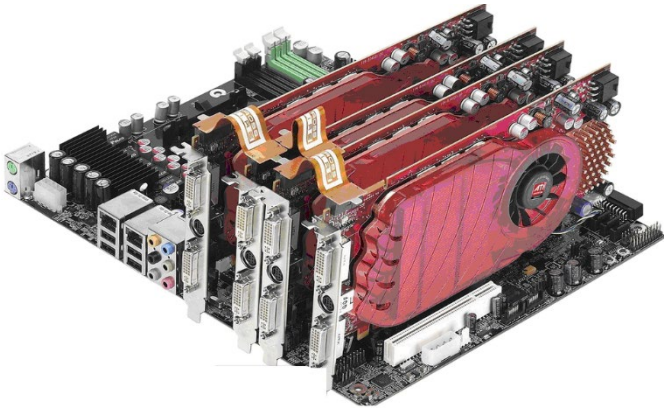

Figure 7 – AMD CrossFire

Source: https://www.dignited.com/58471/nvidia-sli-vs-amd-crossfire/

Choosing between SLI and CrossFire depends on your needs, budget, and the tasks you are going to do. If high performance and optimization are important to you, SLI may be a better option. If you're looking for a cost-effective solution, CrossFire might be a good choice.

CrossFire works with APUs (integrated graphics units in processors) and discrete graphics cards through what AMD calls hybrid graphics, formerly known as Hybrid Crossfire, and Hybrid CrossFireX. The technology is mainly aimed at lower-end cards and for notebooks, where the integrated graphics unit and discrete graphics processor join forces to improve the graphics capabilities of the system, but also to manage power consumption. Nvidia requires motherboards to be SLI-certified, which means

manufacturers must pay for a license and PCIe slots must meet a set of specifications. On the other hand, CrossFire compatibility is much less strict, and all the motherboard needs is a pair of PCIe slots. Therefore, motherboards with CrossFire support are more accessible and have a much lower cost.

Let's consider algorithms for building images when using several video cards. Scissor – dividing the screen into several non-intersecting zones (Figure. 8). In mode Scissor the upper part of the frame is displayed by one map, and the lower part by another. The border of the zones does not necessarily have to pass in the middle of the frame and can be chosen dynamically, based on the complexity of this or that part of the image – that is, there may be fewer objects in the upper half than in the lower half, and then one of the accelerators will be idle, which can be compensated by increasing his area of responsibility. Tasks of such dynamic balancing are non-trivial and require scene analysis, which is not always convenient.

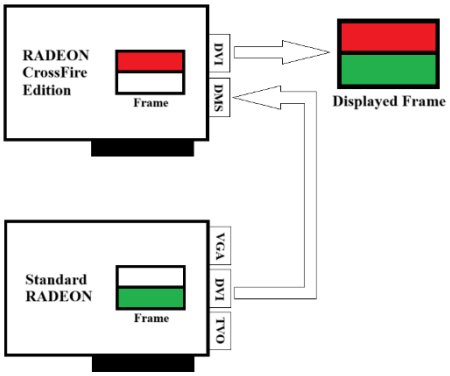

Figure 8 – Scissor mode

Disadvantages of the method that it requires balancing of zones for uniform load distribution; there may be problems with smoothing at the junction of zones; requires significant intervention in the driver and therefore a high probability of unexpected and incorrect operation of some applications.

The SuperTiling (Figure. 9) mode involves a checkerboard alternation of the calculated pixels in this mode, the frame is divided into blocks (also called “quads”). Even blocks are calculated by one card, and odd blocks by another.

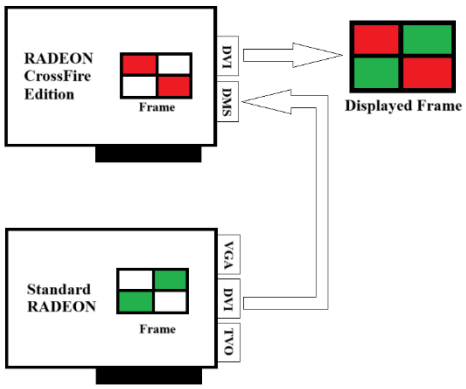

Figure 9 – SuperTiling mode

In general, with the help of this mode, you can display all applications and games. AMD has set this mode as the default for 3D games. Despite each map displaying half of the image, the geometry must be fully calculated on both maps. As a result, productivity growth is not what it could be. However, even with such shortcomings, modern and future games that heavily use pixel shaders will give excellent performance in this mode. In general, the more the game uses shaders, the more there will be a gain in this CrossFire mode.

Disadvantages of the method: does not divide the geometric load and therefore requires a significant margin in geometric productivity; requires sufficiently synchronous operation of video cards and, accordingly, their maximum identity.

The mode of alternating frames that are calculated is called Alternate Frame Rendering (AFR). The AFR mode is the fastest – in it, the cards output frames one at a time (Figure. 10). Let's say the first card displays all the odd scenes, and the second – all the even scenes. Unlike SuperTiling and Scissor modes, where both cards must calculate the geometry for each scene, AFR mode allows each card to do only half the work, because each card calculates geometry and shading only for its frames.

Disadvantages of the method: uneven staff rotation and load distribution; performance is highly dependent on the CPU and system, as well as the nature of the scene, and drops with increasing frames per second; there may be a delay between the frame being displayed and the frame currently being built.

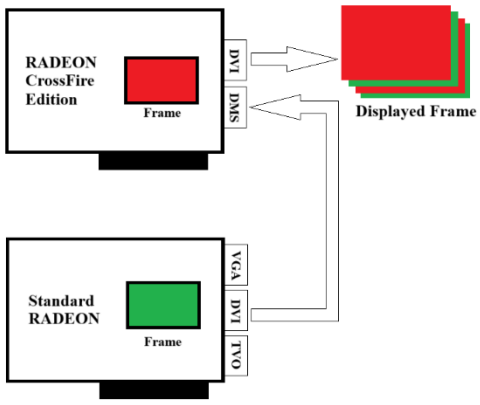

Figure 10 – Alternate Frame Rendering mode

The last disadvantage of this method does not allow it to gain global recognition of users and to some extent inhibits the spread of the entire multi-GPU technology in general (and both SLI and CrossFire). The fact is that this method of displaying the image sometimes gives the effect of "microlags" – small jerks in games that do not depend on the load and speed.

Summary and conclusions.

Since 3D image rendering is a labor-intensive process that significantly impacts the time needed to generate graphical scenes, the issue of parallelizing the computational process becomes highly relevant. Three approaches to parallelizing the process of generating realistic images are identified: functional, spatial, and structural. The use of each approach is determined by the application area as well as the tasks at hand. The use of multiprocessor implementation allows for the superposition of the graphical scene and the independent generation of different areas of the scene.