Lappeenranta University of Technology
Ti5317000 Parallel Computation

Seminar work

Using parallel computing in rendering 3D-animations

Toni Helenius
Teemu Kinnunen

Table of Contents

1 Introduction

Scientists, animators and people who are in touch visualizing items using 3D-modelers, always wants better quality in less time. Computing power has been increased a lot from when first animation was rendered using computer. However, need of computing power has been increased more than computing power. Though, researchers have been finding new technologies to extend computing limits to provide computing power for rendering 3d-animations. Increased computing power is used for creating more authentic computer graphics.

Animators are using a lot of parallel rendering while they are rendering their final product. Every second needs 25-30 frames to be rendered. So movie which lasts for one hour has 9000 – 10 800 frames. And it can take more than hour to render one frame of the movie by using single normal desktop machine. Of course it depends on how complex is the scene and how fast computer is. Though, animation companies such as Pixar or Disney are having lot of thinking how to decrease time used in rendering and get better quality of picture.

So there is definitely need for extra computing power for rendering. People have tried to go round of limits of one computer by using several computers and running rendering task in parallel. The problem is how to divide work to several computers and how computers should change information between each other. Researchers has invented various of ways to divide work to sever processing elements. Division of work, in parallel rendering case, is more state of art than general way than general issue. That is why we are concentrating on those things in this paper.

2 Basics

2.1 OpenGL and DirectX in brief

There are two competing standard specifications defining APIs for writing application for rendering 3D and 2D computer graphics, OpenGL (**Open G**raphics **L**ibrary) and DirectX. In all simplicity these APIs are used to hide the complex nature of different 3D accelerator cards' interfaces, providing the coder a simple uniform interface. [14], [15]

DirectX is Microsoft's standard, first released with Windows 95 in late 1995. It is a series of libraries which, besides the graphical functionality (Direct3D), offer APIs for sounds, networking and controls. Basically everything you need in a 3D application. DirectX only works in Windows operating system which efficiently limits its usage. It has come a long way from a simple API library, now running in 10th revision working only in the latest Windows operating system, Windows Vista, and clearly dominating the PC gaming indrusty. [15]

OpenGL is an open source standard offering over 250 different function calls which can be used to form complex 3D scenes from simple primitives. OpenGL and DirectX mainly share the same capabilities, though OpenGL may acquire these slightly slower than its rival. But the fact that OpenGL is truly open and cross-platform API, it has solely conquered the industrial sector applications like CAD, virtual reality and scientific visualization. [14]

3 Subdivision of work

3.1 Parallel Rendering

When we are wanting to render one frame at time parallel, we need to divide frame in parts, which are rendered in different processing elements. There is couple of different ways to divide dataset to processing elements. Take a look figure 1: Parallel rendering architectures. [1], [2]

3.1.1 Sort-first

All primitives are sorted before they are going to be rasterized. After primitives are sorted they are sent to processing elements. Processing elements rasterizes primitives and after all primitives are rasterized, frame is composed from primitives. If primitive moves to different location then processing elements needs to send primitive's information to another processing elements. And if there is, and usually there is, lot of primitives and they are moving, it causes lot of traffic between processing elements. [1], [2]

3.1.2 Sort-last

In sort-first method we are first dividing each frame in parts. So frame consist of several parts. Each part is then rendered separately in different processing element. Some of the parts might have more elements than another have. It is bad, because then some of the processing elements needs to compute more than another. If some parts of the frame are solid black and somewhere is transparent object (e.g.

glass), times to calculate result might differ a lot. [1], [2]

3.1.3 Hybrid sort-first and sort-last

In hybrid architecture we are using both of the previous architectures. First we are using sort-last method to divide object to different render nodes. Then nodes renders their objects and gets full size image where is that object rendered. Then it divides image like in sort-first architecture and sends parts to different processing elements which are compositing those images. After compositing nodes have gathered and composited all of their images together, they sen parts to display node which gathers parts of the frame together and displays hole frame. [1]

This hybrid sort-first and sort-last architecture is quite used, because it has all the advantages of sort-first and sort-last architectures. [1]

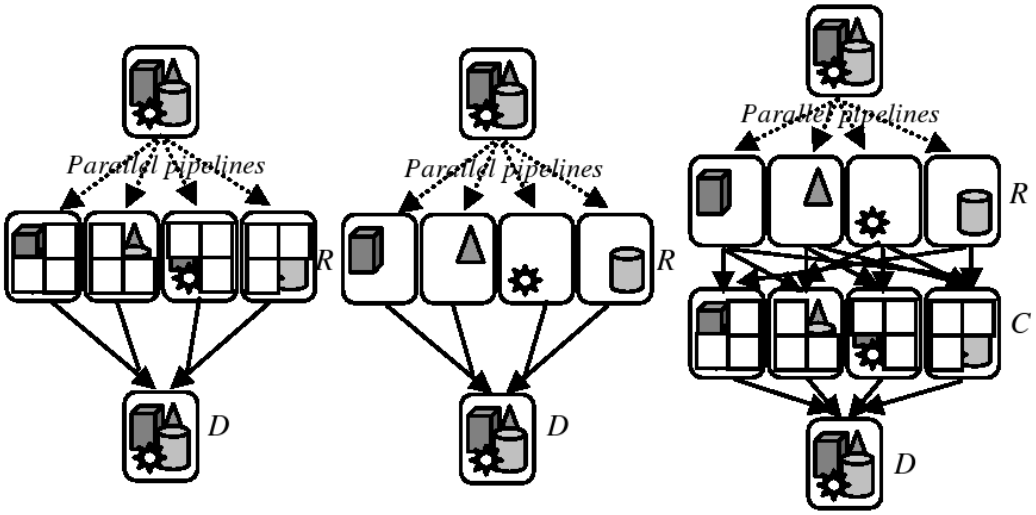

Figure 1: Parallel rendering architectures [1]

3.2 Traditional parallel rendering

We can also distribute work for several processing elements by giving each element different frame. It is very easy to implement, but not quite effective if number of frames is low or if different frames are

depends from previous frames. This is not optimal solution because rendering needs often large amounts of memory. And if processing element can't store everything in RAM, it has to read often information from disk which is very slow.

4 Applications

4.1 Blender3D

Blender3d is open source 3D-modeling tool. It also includes built-in render, game engine and other tools. But we concentrating on render application. It is also possible to use another render with blender. One very “famous” alternative is Yafray. It is contributed from Pov-Ray and it specialized on light effects like ray-tracing and illumination. [3], [5]

4.1.1 Blender3D built-in render engine

4.1.1.1 Most important features (especially for parallel computing)

- Fast
- Oversampling, motion blur, post-production effects, fields, non-square pixels
- Tile-based and fully threaded
- Render Layers and passes
- Render engine tightly integrated with the node compositor
- Vector motion-blur post-process effect (using node compositor)

[3]

Blender3d's built-in render engine is using sort-first architecture. Is dividing frame in smaller parts and each part is rendered in own thread. This makes engine very parallelized, because the most computation time needing operations are executed in threads. [4] It is also possible to render parallel using external tools such as reppu or build-in render daemon. [3]

4.1.1.2 Render pipeline flow

Blender's render pipeline consist of several steps. See figure 7.1: Blender render engine pipeline. If we

are rendering animation we will start from ANIM-state else we start from RENDER-state and render just one frame (image). In both cases we will first initialize render handler. Scene stores render data settings. [4]

Next step for animation-case is to initialize movie. It means that we are preparing to render multiple frames. So, we are doing render-case multiple times and after it is ready we gather all information in the one body. [4]

After that we are ready to do rendering, if necessary. Also, if we are rendering animation we need render instance. It is used to render different frame. It also helps to render animations parallel, because every frame is rendered using different render-instance. In this step we are actually rendering parts of the frame and after all parts are rendered we are compositing those parts to one solid frame. This step is called do_render_composite_fields_blur_3d(). [4]

Previous step (do_render_composite_fields_blur_3d()) consist four sub steps: do_render_fields_blur_3d(), do_render_fields_3d(), do_render_blur() and do_render(). In first step we are checking render options. If field or blur options are set to be executed then those steps (do_render_fields_blur_3d() and do_render_fields_3d()) are going to be performed. After those steps are performed then rest of the image is going to be rendered. [4]

After all frames renderer, we are storing result in the specified location and possible play/show result. [4]

4.1.1.3 Render pipeline API

See Appendix 7.2.

1. Initialize or get new render instance.
2. Initialize state: gets render data, view plane, display space and initializes result buffer(s).

3. Set camera view.
4. Callbacks. Stop rendering if something is wrong.
5. Create render data, shadow buffers, environment maps and octree.
6. Render loop. This phase is possible to parallel with multiple processors. See Tile processor.
7. Free render data.
8. Save and show result.

[4]

4.1.1.4 Tile processor

Tile processor is starting threads which are doing executing rendering function. Frame is assembled from many tiles. Tile processor is rendering those tiles. Each part is rendered in different thread. [4]

In first step, tile processor is initializing parts which needs to be rendered. So it checks how many parts is needed to be rendered and allocates memory for those partitions. Also it starts threads. [4]

Next step it finds first part to render. And if there is parts still to be rendered in enters to rendering loop. [4]

Then it checks if there is thread slot available, it sets rendering task to that slot. Then thread begins to render that part. After rendering is done it finds new part to be rendered. After part is rendered, thread is removed and result is showed. [4]

After every part of the frame is rendered, result is stored and threads are stopped. [4]

4.1.2 Yafray render engine

Yafray based on Pov-Ray rendering engine. It is stand-alone render and it can be used with various of modelling tools like Blender3d. It is multi threaded so it can render animations and images parallel. It is specialized in light effects like ray tracing, global illumination etc. [5]

4.2 Chromium

Chomium is a cross-platform open-source project for interactive rendering using clusters of workstations. Some may remember a Stanford University project called WireGL, Chromium is a follow-up to it; far surpassing the WireGL's features. Chromium allows filtering and manipulation, such as parallelization, of OpenGL command streams. Usually parallelization of code is a difficult task, while Chromium dynamically replaces the system's native OpenGL libraries with its own when the application is executed; it promises that OpenGL applications can be used without modification with Chromium. But of course the maximum benefits can only be achieved by writing Chromium specific code, achieving parallelization through special synchronization primitives. [7]

Chromium is a highly scalable system. It can use any number of rendering nodes to calculate pixels, can handle huge amount of data (huge images) and can divide the result to any number of display devices. It can be virtually applied to any size of cluster! But its capabilities are not only limited to clusters, the technologies are also useful in single desktop PC. It supports Sort-first, Sort-last rendering described earlier, as well as hybrid sort-first and sort-last methods. [7]

4.2.1 Applications: The NCSA TerraServer Blaster

TerraServer is one of the world's largest online databases resident at http://www.terraservice.net/ for everybody to see. Database roughly 3.3 terabytes in size, holds vast amounts of aerial photography and maps of the whole USA. [8]

The NCSA (National Center for Supercomputing Applications) TerraServer Blaster is an OpenGL interface to the TerraServer. It can be used with Chomium to render huge areas to large resolutions, to the NCSA display wall for example. [8], [9]

The Wall project at NCSA uses 40 (8x5) LCD projectors, each cabable of displaying 1024x768 resolution images, to create a single image with resolution of 8192x3840! The system consists of 49 node high performance graphics cluster; 1 head node, 40 display nodes (1 for each projector) and 8 I/O

nodes (Figure 2). These Linux nodes are equipped with NVIDIA's GeForce 5900 FX graphic cards. Connection is handled with Myrinet (gigabit Ethernet). Chromium distributes the work on the cluster. The result: huge, incredibly detailed map on the wall ready to be studied! [9]

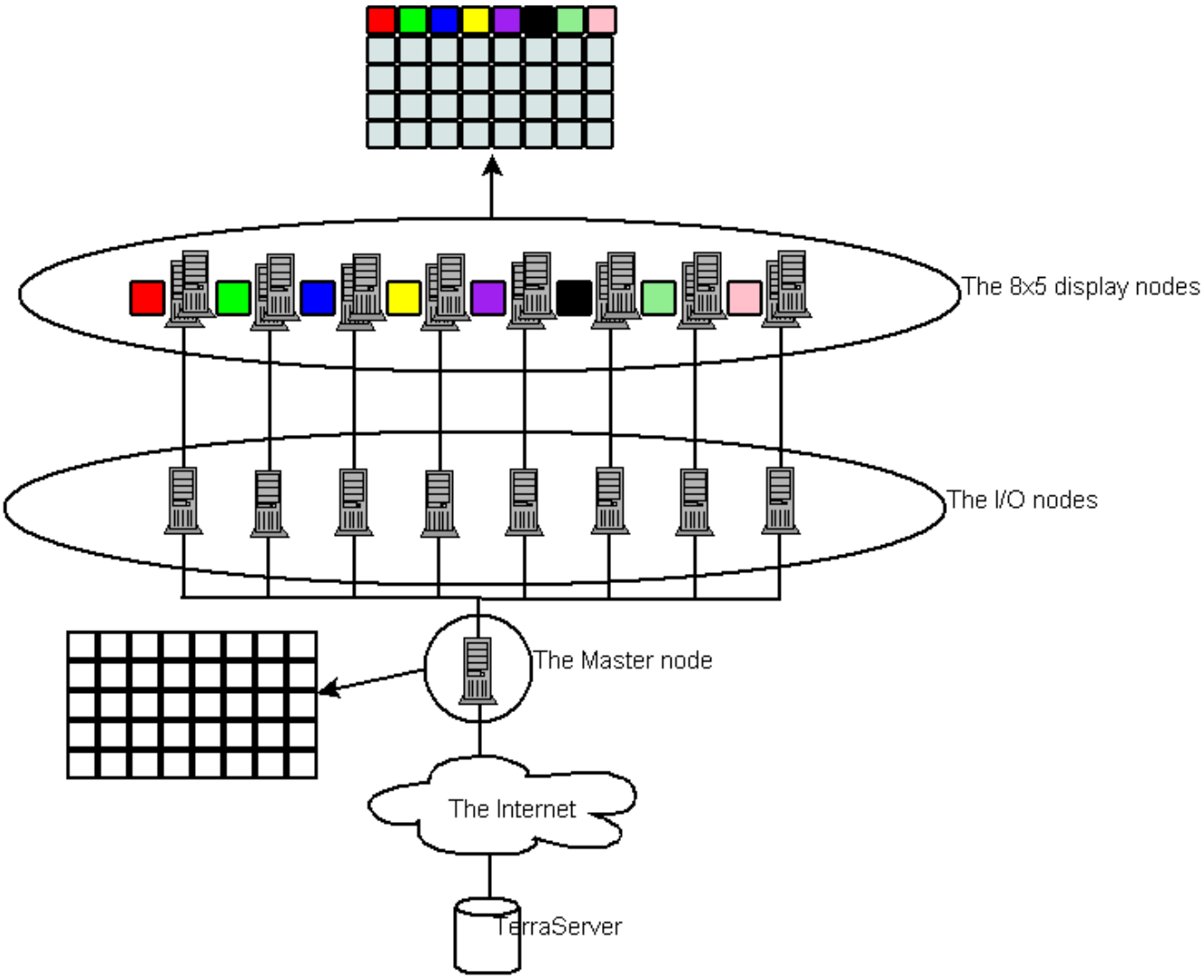

Figure 2: Approximation of the Wall project layout

Note that figure 2 illustrates a possible system with the given specification, just to give an idea of the system complexity. The actual Wall project cluster was applied in 5 tidy closet size cases! [9]

There were no clear benchmarks available, but the project members claimed playing some First Person

Shooter games with the 8192x3840 resolution. Especially Chromium modified Quake III by ID Software. Needless to say that such game requires high FPS (Frames Per Second) to be playable so we can only presume that the system is capable of providing real time rendering in high resolutions. However it is impossible to test such resolution in my home computer to get any comparison. [9]

The NCSA TerraServer project is used in other systems too to create similar large display systems that use a cluster to render huge resolution images. Most mentionable might be the Boeing Company. [9]

5 Hardware

5.1 CPUs

While more efficient processors are more efficient of course, but how about if we don't want to use all the capacity? If we just need lots and lots of small tasks such as small pictures that will someday make a bigger one! This kind of parallelization requires certain kick from the hardware. The solution could be multiple processors or, what seems to be more popular nowadays and cheaper too, to have several cores on single processor die. Latter is cheaper and the I/O between the CPUs is clearly faster.

Dual core CPUs are common ground and quad core few steps away from present. However it is to be noted that there is very little or nothing to be gained in sense of a single application if it doesn't even try to run in parallel.

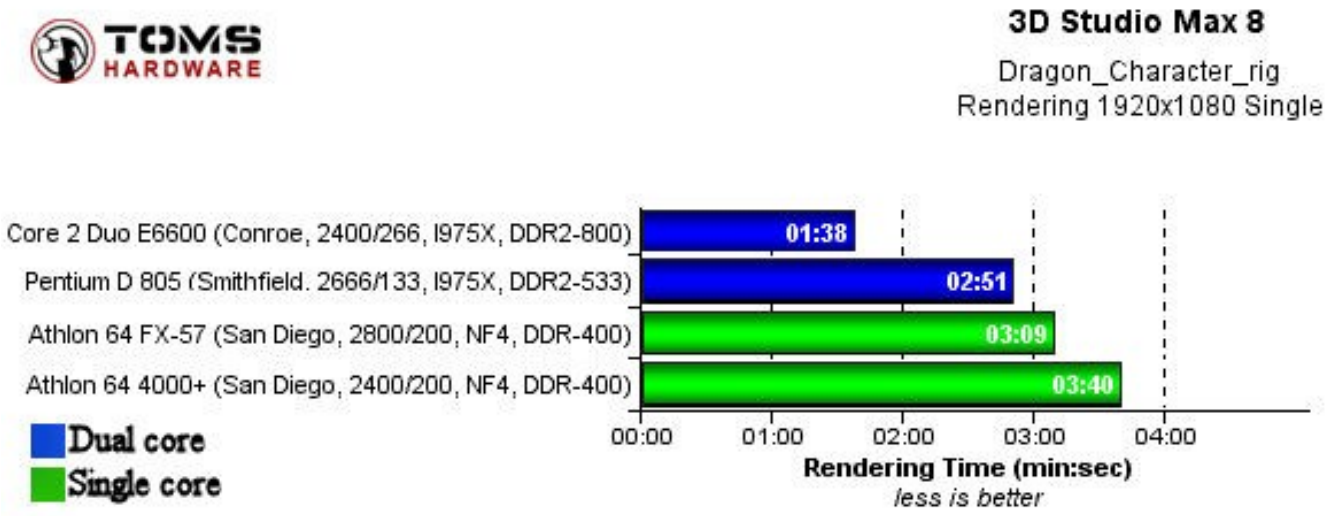

Figure 3: Benchmark between dual & single core rendering with 3D Studio Max

Figure 3 tries to show us some benefits from dual core processors compared to the single core ones. Some gains are explained by the newer technology and faster memory. But AMD's Athlon 64s did offer superior performance compared to Intel's Netburst technology which is still used in the Intel Pentium D 805, the slowest and cheapest dual core. And it still beats clearly the high cost and performance level AMD single cores. The modern Core 2 Duo is offered as reference, but it's no way directly comparable to these Athlons in sense of core count. [10]

5.2 GPUs

GPUs are favored in real time graphics, unlike many of the other rendering needs are satisfied by the CPU. Newest models of GPUs can be also used for general computing, it is called GPGPU's. GPGPUs are very fast in certain kind of problems. And because they are developed for creating graphics, they can be used for rendering animation also. Good example of this is Nvidia's Gelato render engine. [11], [16]

As well as in CPUs the trend in GPUs is similar. Enthusiastics find themselves with SLI (NVIDIA) or CrossFire (ATI), two graphic cards connected together with cable. This again requires specifically optimized applications to take full advantage. These solutions are highly favored in modern PC gaming. However gains from such systems are at maximum 1.5 times the power of the cards added together, but

in practice usually as low as 1.2 (figure 4). So if you must harness this power, you can, but you'll see the results more in your wallet than in your computer. [10], [11], [12]

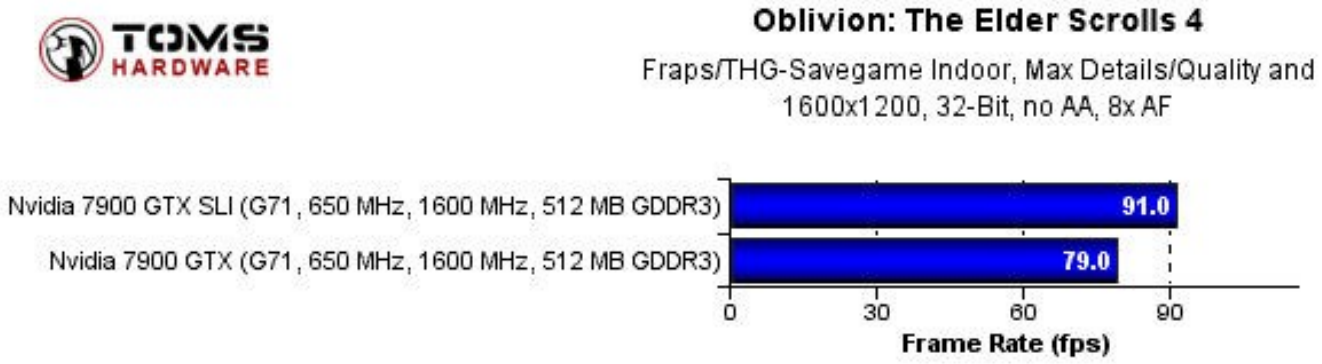

Figure 4: SLI and single card benchmark

It's difficult to see if this gets any wind beneath its wings; there is at least one dual GPU graphics card. 2 GPUs on one card significantly reduces the latency introduced in SLI or CrossFire technology. While they are increasingly complex, compared to modern CPUs they already might have more transistors. And the demand for fast, and faster, GPUs is enormous, especially in gaming. They hardly parallelize by themselves and that may lead to a stop in performance. The heat dissipation and power consumption issue might cause a pause in this sector. We are not quite yet there when it comes to GPUs… [10]

Current graphic cards are highly parallelized, they have several processing elements in each card. Also those processing elements are specialized for doing their task fast as possible. Fragment and vertex processors are having their own task, where they are good. When we are using two graphic cards together, the bottleneck is usually processor or memory bandwidth between processors or graphic cards. [10], [11], [12]

6 Test case: Blender3D with Elephants Dream

6.1 Test setup

For the test we acquired a ready scene from a well known project; Elephants Dream, available at http://www.elephantsdream.org/. It is a massive open-source project whose goal was to create a short

animated film using only open-source tools. The source code is available in Blender3D code. After nearly a year of work it was finally done in May 2006. Total running time of the movie is 10:53. For the test we rendered the first 10 frames from the prequel of the film. 10 frames are equal to about 0.3s of video! [13]

The test was made with a brand new computer with Intel's Core Duo 2 processor which offers 2 logical cores running in 2,4GHz on one physical die. The idea was to test how well our example application Blender scales on a modern workstation with multiple CPU cores.

6.1.1 Detailed specification of the testbed:

6.1.1.1 Computer:

- Intel E6600 Core Duo 2 processor
- 2GB 800MHz Dual channel DDR2 RAM
- Intel DG965WH motherboard
- NVIDIA GeForce 7900GS 512MB graphics card

6.1.1.2 Software:

- Microsoft Windows XP Professional Service Pack 2
- Blender3D version 2.42a

6.1.1.3 Settings:

- Elephants Dream file: 01_02.blend
- Rendered 50% of the original size
- Each frame has set to have 36 individual peaces (tasks)

6.1.2 Executing the test

The test was done in 2 phases. Blender3D allows rendering with 1 or 2 threads. First phase was done with 2 threads running and the second of course with just one thread. We monitored the CPU utilization of the whole process to see how it scales and how optimized it is. To monitor the CPU usage we used simply Windows' Task Manager. It was accurate enough to carry out this test. A snapshot of about 2 minutes of CPU utilization was measured. On average that renders 1-2 frames on the test setup.

We chose to divide each frame to 36 little peaces (6x6), the default was 16 (4x4) after testing various rendering options with single renders. Just to even up things!

6.1.3 Results

The results were frankly what we expected. Running with 2 threads was a clear winner, it finished over 10 minutes faster than its counterpart, making it about 61% faster in comparison. See the figures bellow.

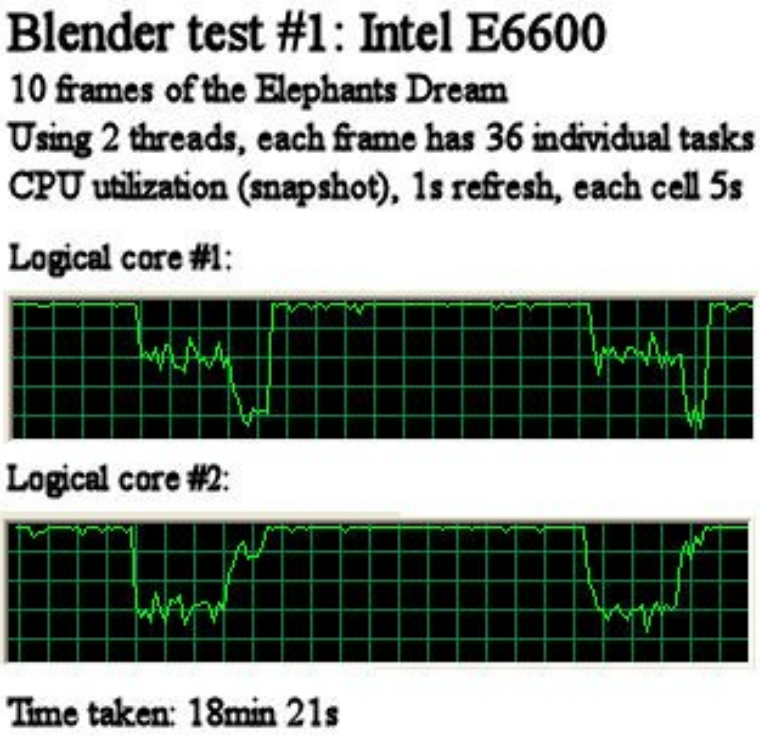

Figure 5: Results of the rendering using 2 threads

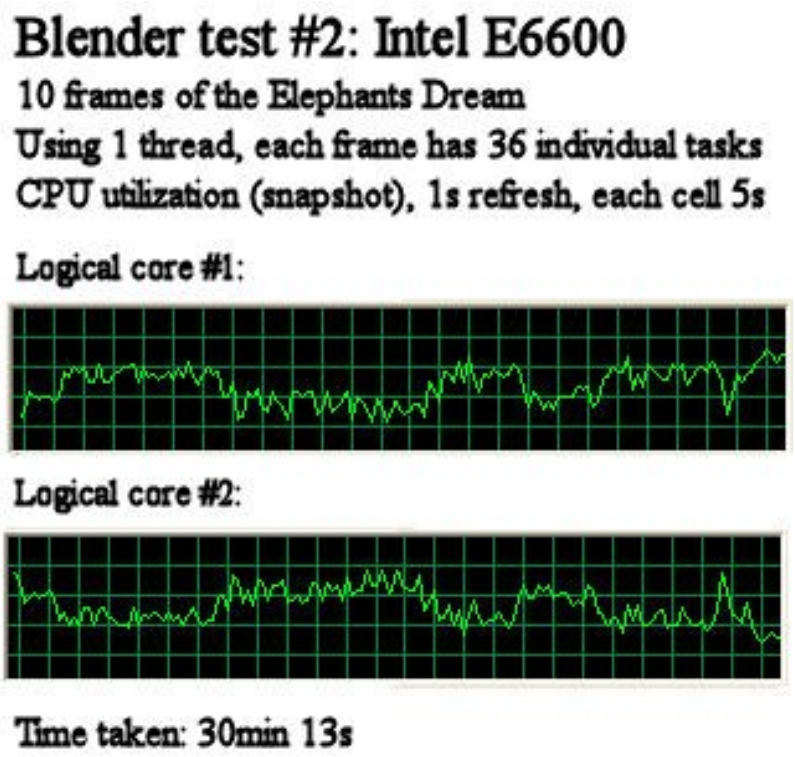

Figure 6: Results of the rendering using 1 thread.

On test #1 (figure 5) the average CPU utilization was somewhere near 99%, that indicates pretty good use of resources. Average RAM comsumption was 350MB. The spikes, or downfalls, on the CPU usage indicate change of frame to be rendered. It falls more radically with core #1, this means it waits for the second thread (assumingly running in core #2) to finish its task, the last of the 36 peaces. After that it takes few seconds to calculate and divide the next frame, and then we go again with nearly 100% CPU usage! Dividing the image to more tasks may even out the downfalls, but the again the tasks are smaller and that causes more downfalls. This division can save a lot of time, but finding the sweet spot for certain renderings may take awhile. With more complex renderitions the division should be greater than with simpler tasks.

On test #2 (figure 6) the CPU usage was quite even throughout the process, by average 52% of the CPU was used. The 2 logical cores battled for CPU time all a long, and as excepted there was not enough work for each. RAM usage was about 300MB in average, little less than running with 2 threads. The rendering took quite a long time in comparison.

The test clearly shows the advantages of parallelism. In large and long animations even a percent faster rendering per frame saves huge amounts of time in the long rung. And it's nowadays very common to even cheapest computer available to have a dual core. And looking to near future, the quad core processors are already available for the enthuastics. It's not too far around the corner for the cheapest processor available to have 4 cores! That makes running in 1 thread a complete waste of time!

7 Conclusions / Summary

In parallel rendering case, limitation of single processing element has been successfully rounded by using parallel computation. There is couple of different ways to divide work between processing elements, but most used architecture is hybrid sort-first and sort-last, according to literature [1].

Blender3d's built-in render is quite fast and able to use threads. It can be also used in cluster or grid by using external applications. Yafray is stand-alone render, at it can be also used parallel. [3], [4], [5], [6]

Chromium gives great framework for parallel rendering. It is very scalable and it can be used also for interactive parallel rendering like for games. It is also quite popular and researchers has been written couple of papers about Chromium. [7], [8]

Benefit from using GPUs beside of CPUs depends on how well program can get benefit of GPUs several processing elements. Also when there is two graphic cards added together using SLI or CrossFire can achieve great benefit from increased computation power if program is able to get benefit from that. So GPUs can provide great benefits if software is developed to getting benefit from GPUs. [16]

Blender3d test case shows that parallel rendering has came to stay. It gives great performance addition for rendering animations and special effects. By using threads we can significantly decrease computing time (if we are having multiple processing elements). Test also shows that it dividing frame in parts is effective way to balance load to multiple processing elements.

8 Appendixes

8.1 Blender pipeline

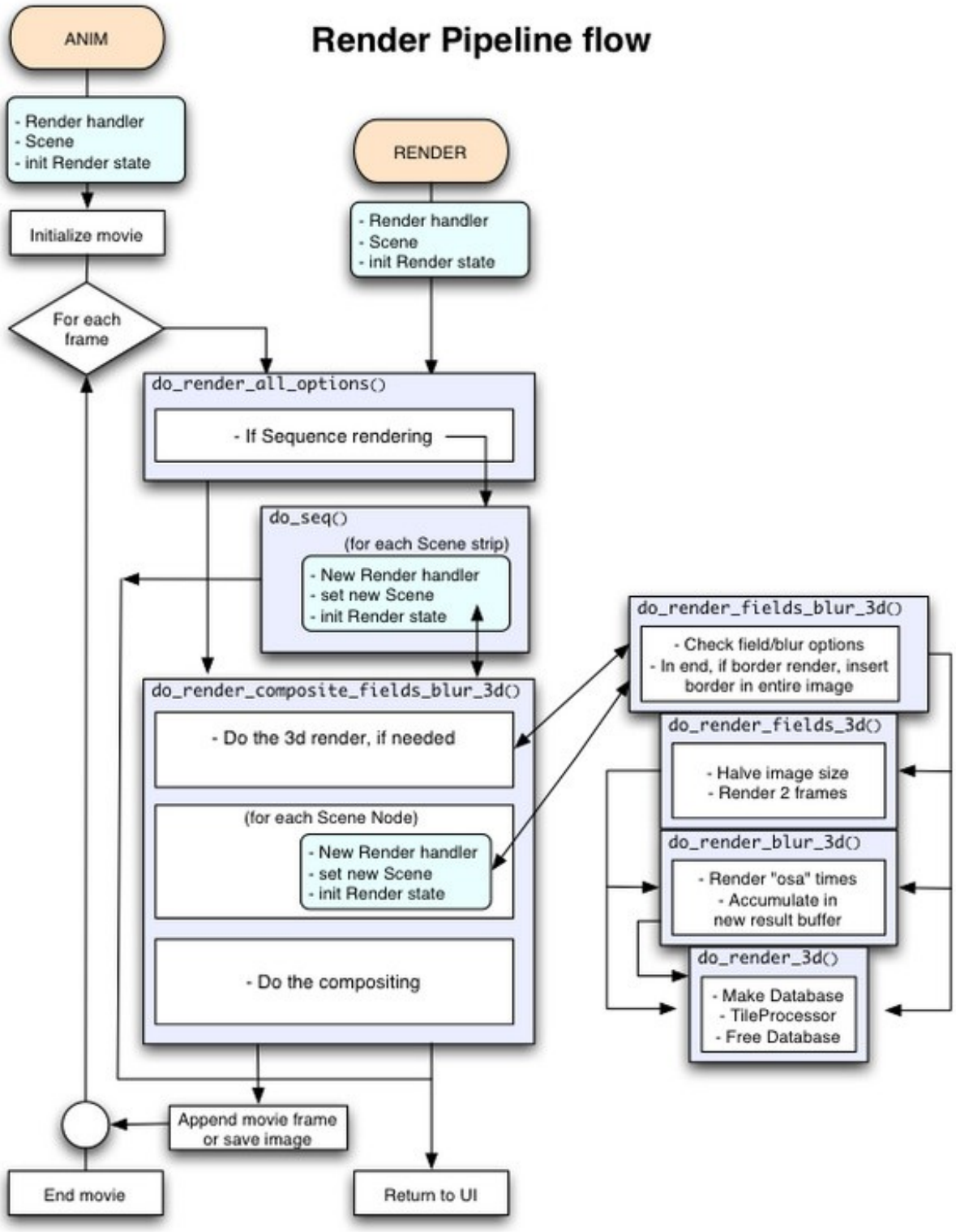

Figure 7: Blender built-in render engine survey [4]

8.2 Blender Pipeline API

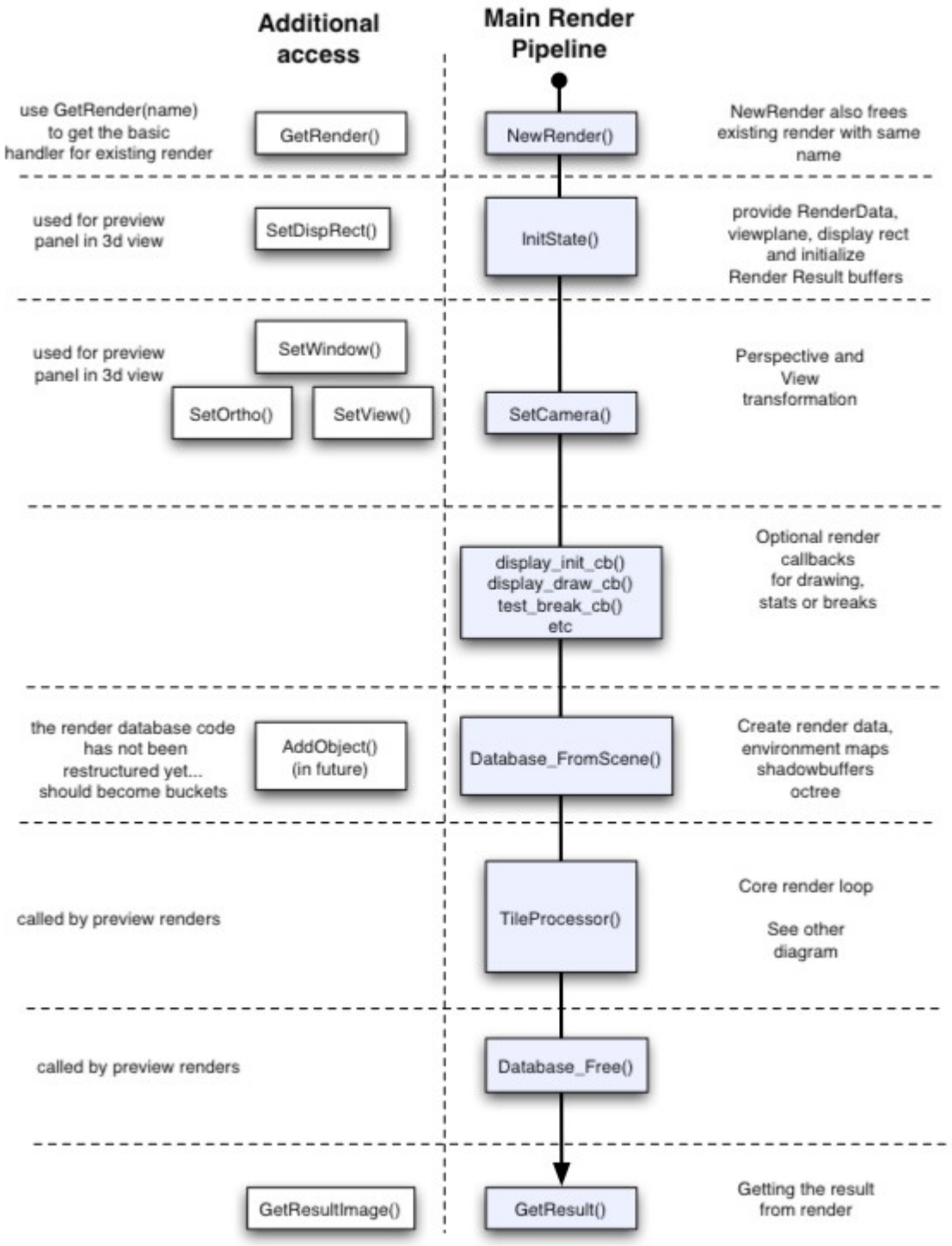

Figure 8: Blender built-in render engines pipeline [4]

8.3 Blender tile processor

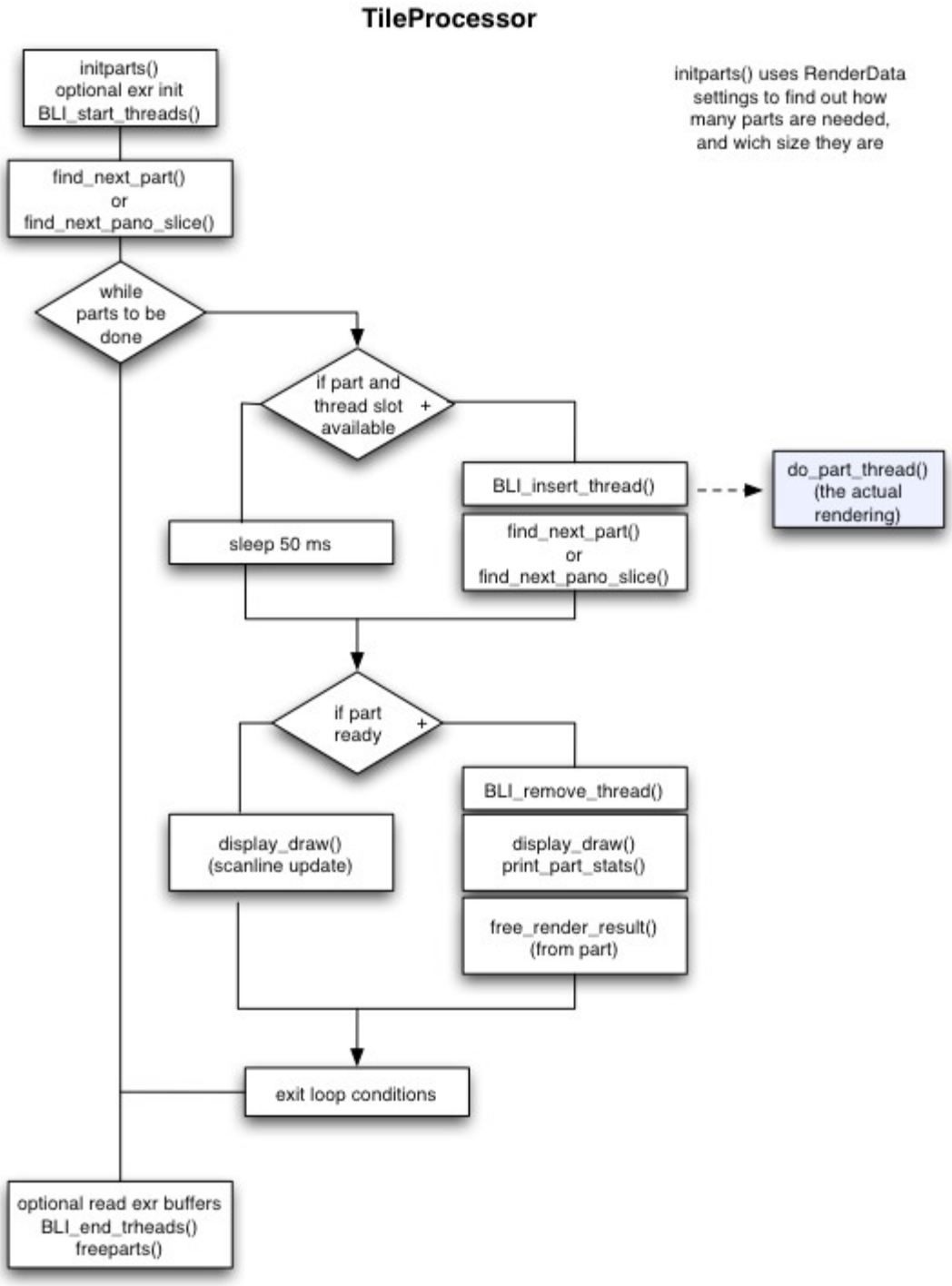

Figure 9: Blender built-in render engine's tile processor [4]

9 References

1. Haoyu Peng, Hua Xiong, Jiaoying Shi, **Parallel-SG: research of parallel graphics rendering system on PC-Cluster,** Virtual Reality Continuum And Its Applications, Pages: 27-33, 2006
2. E. Wes Bethela, Greg Humphreysb, Brian Paulc, J. Dean Bredersond**, Sort-First, Distributed Memory Parallel Visualization and Rendering,** IEEE Symposium on Parallel and Large-Data Visualization and Graphics 2003, 2003
3. Blender3d: http://www.blender3d.com
4. Blender3d render engine: http://mediawiki.blender.org/index.php/BlenderDev/RenderPipeline
5. Yafray homepage: http://www.yafray.org/
6. Reppu homepage: http://www.imnetti.fi/~desaster/reppu.php3
7. Chromium: http://chromium.sourceforge.net/
8. TerraService: http://www.terraservice.net/
9. The TerraServer Blaster: http://brighton.ncsa.uiuc.edu/~prajlich/wall/tsb.html
10. Tom's Hardware: http://www.tomshardware.com/
11. NVIDIA: http://www.nvidia.com
12. ATI: http://www.ati.com
13. Elephants Dream: http://www.elephantsdream.org/
14. OpenGL homepage: http://www.opengl.org/
15. DirectX homepage: http://www.microsoft.com/windows/directx/default.mspx
16. Nvidia's Gelato: http://www.nvidia.com/page/gz_learn.html