
Production Systems and Information Engineering
Volume 10 (1), pp. 16–29
doi: 10.32968/psaie.2022.1.2.

THE PAST AND THE FUTURE OF COMPUTER VISUALIZATION

Péter Mileff
University of Miskolc, Hungary
Department of Information Engineering
mileff@iit.uni-miskolc.hu

Judit Dudra
Bay Zoltán Nonprofit Ltd. for Applied Research, Hungary
Department of Structural Integrity and Production Technologies
judit.dudra@bayzoltan.hu

Abstract. Computer visualization has a long history with many great results. However the 21th century has seen the greatest progress in development. The area continuously followed the evolution of the available hardwares. As more and more computing capacity based computers became available, computer visualization had been unfolded. New algorithms and technologies have appeared. Nowadays, there is no area in our daily lives where the need for visualization does not appear, whether it be a car, a computer game, or even an IoT device. This paper examines and makes a brief overview abount the dominant trends that modern real-time visualization currently follows. Key technologies are covered that are an integral part of today's rendering and the opportunities that could be decisive for future real-time rendering. It is difficult to predict the future of visualization and the preferred technology by the developers, as it depends heavily on what hardware vendor-supported technology becomes widely accepted and dominant.

Keywords: computer visualization, voxel based rasterization, software rendering, ray-tracing

1. Introduction

The beginning of computer visualization can be traced back to the time of the first computers. The availability of hardware as a natural need has led to the development of the area. Visualization has now become an integral part of our lives. There is almost no area where it has not appeared in some form. Any application that collects data may require visualization. We can even think of the dashboard and auxiliary displays of modern cars here. There are areas,

where the application development can only be done visually. Examples are a computer game, a cartoon, or even a CAD model of a part. There are no ads or TV commercials that do not include any visualization. The human is a visual type. Whatever we need to demonstrate, understand or comprehend, we prefer to use illustrations, animations and other graphics to increase intelligibility.

The most important goal of computer visualization is to model reality as accurately as possible with the available hardware, which is a very complex, multifactorial task. Based on these, two important types of rendering can be distinguished: *real-time rendering* and so-called *offline rendering*. The two groups have been separated from the beginning. As their needs are different, their development has taken different forms.

Offline rendering is designed to achieve high image quality. Its typical application areas are the rendering of frames for (animated) films, three-dimensional models (e.g. car, building, etc.) taken from a given view. Since the main goal is to reach the highest image quality and the best possible approximation of reality, the calculation of pixels is not performed in real time.

The calculation of the frame without a strict time limit makes it possible to take into account the thermal interest factor that determines a realistic representation accurately and physically correctly. Typically, such factors are the efficient handling of light, illumination model, reflection, surface material properties, and shadows. In these forms of representation, the really good result is actually the use of the so-called *global illumination*, where the (necessary) indirect lights are also modeled and calculated.

The field of offline visualization has evolved a lot over the years, but not as much as real-time visualization. Most of the algorithms that try to approximate reality correctly were already available in the very early days, but the limitation was the hardware available in each case. With this form of representation, it was recognized from the outset that the calculation of frames could not be left to a single computer. Over time, local and now global so-called *render* farms appeared.

Now in modern 3D computer graphics (design) software (e.g. Blender, Autodesk 3ds Max) there is a built-in ability to configure and distribute the rendering calculation of a graphics content between different computers. This can be called a local render farm. Because the rendering process usually has huge computing power requirements and it is not easy to maintain an own computer rendering farm, online render farms have been appeared.

As well as with the spread of broadband internet and the rapid development of web and cloud technologies, online rentable computing capacities are now available to compute the visual elements of a single frame or even an entire

movie. It is important to note that only CPU-based computing capabilities have been available in the area for quite some time, but GPU-based farms are now available. The reason for the proliferation of relatively late GPU solutions is that they often adhere to 64-bit computing to achieve high image quality. GPUs, on the other hand, only offer this in recent years. The other reason, due to the late support, is that we can even divide the calculation of one frame into several machines. However, for GPUs, this was also difficult at first, because driver and API support had to be developed for this.

The article hereinafter focuses on real-time visualization, which can undoubtedly go a long way behind it, and has undergone tremendous development in recent years. A strict condition for real-time rendering is (usually) that at least as many frames per second be calculated as is already continuous to the human eye and that is necessary for the given visualization or interaction. For computer games, this typically means 50 to 60 frames per second. Below this, for certain types of games - where events change faster - the gaming experience is not satisfactory. Here, perhaps, the above-mentioned goal of computer visualization could be supplemented, according to which the most accurate modeling of reality is the most important guideline. In fact, everything needs to be achieved in real-time visualization, evolving to the point where images produced now offline are produced in real time. The development of hardware and all efforts are mostly in this direction, which is why we can say that it is the most developing area. In the following the most important trends and techniques of visualization are reviewed below.

2. Evolution of the hardwares

It is safe to say that the main driving force of computer visualization is the gaming industry. Since the advent of the first computers, there has been a growing demand for playing with the computer in some form. And of course, since visual quality is a key player in the end result, it has undergone a very strong development. In the beginning, it was only possible to play quality games on arcade machines. The real explosion was the advent of home computers at the right price, when it became available to almost anyone to buy at least one desktop computer.

Important players in this age were the 386, 486, 586, 686 and Pentium I. type computers. These hardwares did not have good performance compared to today's computer. Weak CPU, low and slow memory, limited BUS speed (ISA, PCI) and of course a complete lack of hardware-accelerated rendering.

In comparison, an Intel 486DX-33 (1990) at the time had 0.03 GFLOPS performance, while an Intel Core i7-3770 (Ivy Bridge) (April 2012) had 108.8

GFLOPS performance. Despite this (and the lack of internet) the gaming industry began to flourish, this period is called the DOS era.

Despite the low hardware performance, great graphics applications / games were born. The typical language of software development was C, PASCAL, and performance-critical parts were then embedded as assembly blocks. The programs (especially for games) had to be well optimized, as neither a dedicated GPU-based graphics card nor serious performance was available, everything was calculated by the CPU. This was further complicated by the fact that the programming tools and environments available at that time were not as flexible and comfortable as they are today. Perhaps the Pentium I, II, III, and IV families can be mentioned as a separate era. A very important milestone was that the so-called MMX extended instruction set was already available in the central unit of the first-generation Pentium, and later more advanced versions of it were called the SSE family.

Hardware-accelerated GPUs were still not developed at this time, the graphics were still calculated by the CPU. MMX and SSE are a SIMD (single instruction, multiple data) instruction set. They allow the same arithmetic operation to be performed on a large amount of data at a time. SIMD was a major step forward in accelerating calculations, allowing large vectors or matrices to be manipulated in less time. SIMD instructions allow easy parallelization of algorithms used in audio, video, and video processing. In practice, in computer visualization, this meant that the processor was able to perform some transformation on multiple pixels simultaneously. Following its release, the software soon began to support the new family of instructions, even though creating speed-critical parts of a software based on MMX / SSE was not a trivial task. Shortly after the MMX line was born, AMD also came up with an expanded set of instructions, 3DNOW!, which was a similar extension, mainly to help with three-dimensional rendering. Hardware manufacturers soon realized that the rendering process could be well separated from the rest of a piece of software as well as parallelized. In response to this demand, after many developments, the first true dedicated GPUs (3DFX Voodoo I (1996), Voodoo II (1998), GeForce 256 (1999)) were born. The great advantage of these was that they were able to make a significant improvement in image quality and speed.

Although this paper focuses mainly on the PC line, many technological steps have been required to complete these GPUs. Many smaller bigger successful chips and cards were born. Such was the case with the AGA (Advanced Graphics Architecture - 1992) add-on for the Amiga 4000. Video cards began to spread explosively, being available at relatively affordable prices in any store. And the software immediately began to support it through the OpenGL and DirectX APIs that were started to develop at the time. The GPU market

has been growing steadily since then, despite no major technological innovation since then. Initially, video cards had a fixed-function pipeline, and the programmer had little or no say in the rendering process. An important stage in the development was the appearance of programmable pipeline and the shaders. Development has come a long way, with GPUs for portable devices also appearing. It is safe to say that today's GPUs already have very high performance. In addition to visualization, they can also be used for general computational tasks. If we look at the list of the fastest computers registered by the TOP500, many GPU-based computers are at the forefront.

We must not forget that the GPU is not a panacea. Although they are fast, their price has multiplied over the years, and their power consumption has also increased significantly [14]. Observing the market, a very strong marketing / media support can be felt. Today, almost every device gets a GPU, so it goes without saying that we use it for visualization. This is not necessarily right in all cases, because the used hardware usually restricts the applicable rendering technique. However, the industry has forgotten that there is already a good hardware in the computer that can be used to display or complement it: the CPU.

3. Polygon based geometry

Several approaches have emerged to represent shapes in memory, but the most common and dominant object representation today is the polygon-based approach. In this case, the object or model is usually divided into the simplest convex polygon(triangles) during the modeling, and in the process of drawing these elements are rasterized using some type of algorithm.

The performance and nature of the rendering always heavily depends on the rasterization algorithm. Although a number of different solutions have evolved over the years (e.g. raytracing, volume rendering, etc.), GPU manufacturers use the triangle traversal approach during the eal-time rendering as the prevailing process. The reason for the choice is the performance because the implementation allows for significantly faster visualization than, for example, ray-based algorithms. The triangle, as a separate unit, has a particular importance and can be considered practically the atomic unit of real-time representation.

3.1. Triangle traversal

The triangular traversal based approach was already the basis of early computer visualization and is still the dominant visualization solution today. To put it very simply, the essence of the process is as follows. Triangles mapped

to the region of the screen, like any non-intersecting plane polygon, divide the plane into two regions: the inner area, which is finite, and the outer, which is not.

The boundary between the two is formed by the boundaries of the triangle, that is, the edges. To rasterize a triangle, we essentially have to traverse a set of points with some algorithm that is in some sense part of the triangle mapping, or follow some pattern, and calculate which points are inside the triangle [10]. The color of the inner points/pixels can then be determined. The process is essentially a discretization process:

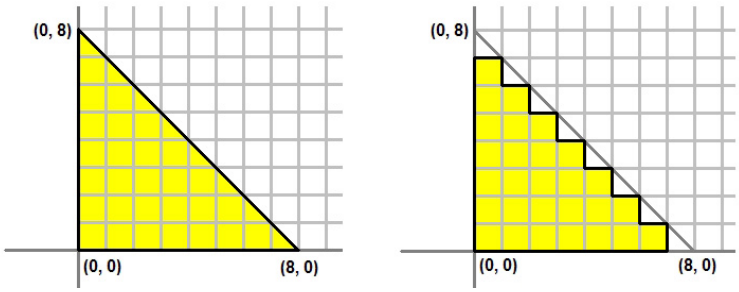

Figure 1. Pixel rendering as a discretization process

In the classical sense, the filling process is performed pixel by pixel so the internal iteration and various calculations have to be performed quite often. Although the process seems simple, the performance of filling, and thus the performance of the application, depends largely on the implemented algorithm, its level of optimization, and the level at which we model reality (lights, reflections, shadows, etc.).

Today, two staining approaches are known that are widespread: scanline and half-space-based algorithms [10]. The basic idea of the scanline approach developed earlier in time is that during rasterization, the triangles are traversed from top to bottom row by row (scanline). Each row represents a line whose start and end are the points of intersection of the sides of the triangle and the scanline along the x-axis. Endpoints can be determined incrementally by calculating the slope value of the edges. The process of filling is essentially the determination of the pixel color values of this line. The half-space-based approach starts from the convexity of polygons: the interior of a convex polygon with n edges can always be described by the intersections of n half-spaces.

Thus, in the case of triangles, the three half-spaces clearly define the range to be filled. By calculating the minimal bounding box of the triangle, using the equation of the three planes and traversing it, the pixels inside the triangle can be filled [10].

Different modified versions of both algorithms can be found in the literature [1]. GPUs use the second approach due to easier parallelization.

3.1.1. Ray based solutions

Another dominant direction in rasterization is ray-based solutions. To name a few: Ray-tracing, Ray-marching, Cone tracing, Beam tracing, etc. These solutions are fundamentally different from the triangle filling-based algorithms.

These approaches work with rays coming from light sources placed in the virtual world. They try to model reality by illuminating spatial objects with photons / rays from light sources while the rays bounce from object to object. With this solution, global illumination can be reached if the movement of a sufficient number of rays is tracked. In practice, it means millions of rays. This is why these methods require a lot of computation, so far they have been used mainly for offline rendering for film and other high-level graphical modeling.

However, today's modern hardware is already powerful, and more and more efforts are being made to use these algorithms in some form, even in limited quality, in real time. The most common problem when making animated films is that designers want to look at a given scene from multiple angles, with multiple light sources with different parameters. And all this without having to wait minutes to render such a scene. The efforts are also visible from the perspective of video card manufacturers. Ray Tracing has never been GPU supported, only in recent years have certain tools (e.g. NVIDIA OptiX ™ API, NVidia RTX based cards) started to appear. And in 2019, Crytek released the first true executable demo application based on real-time ray tracking.

The ray-based approach is a good future direction. As shown in the picture (Figure 2) as an example, more and more exciting ray-based transcripts are in the works. With current hardware, we are almost on the verge of running in real time.

3.2. The future of polygon based methods

More and more news in the field of ray tracing supports the fact that the (near) future of visualization will be based on some kind of ray-based global illumination technique. The question arises as to what is the problem with triangle filling if everyone has supported this approach for so many years. If we look at today's AAA computer games, we can see that they have lavish visuals. However, the quality of this cannot be improved to the extreme. To see a result of this quality on the screen, a lot of little tricks, algorithms and other additional solutions that falsify reality must be used. Such a game engine is already very complex. In the case of a simple (seemingly) point light source,

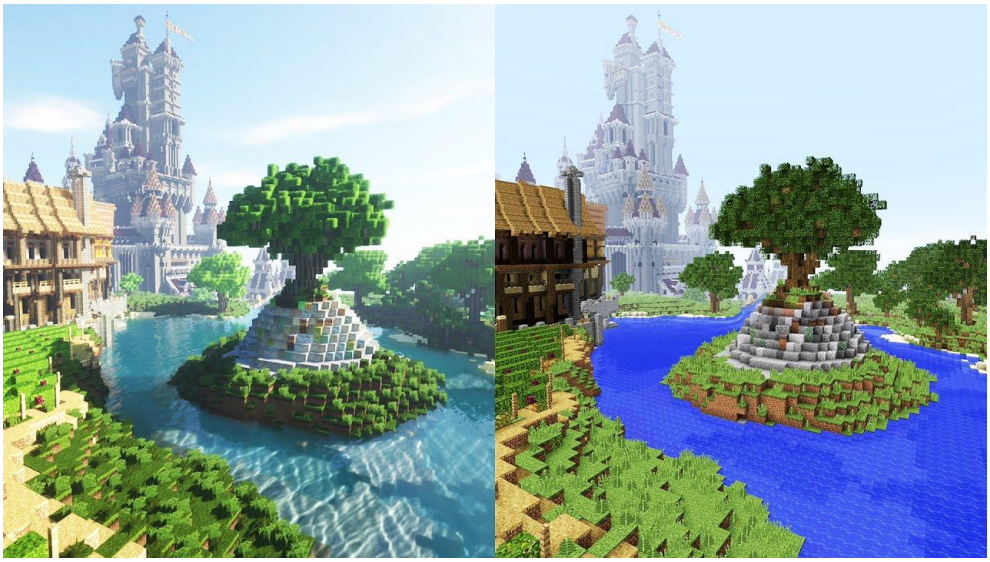

Figure 2. An upcoming version of the famous Minecraft using ray tracing. The quality difference is clear.

four shadow maps have to be created and we have not even talked about their quality.

Reflections, refractions, surface roughnesses are all part of a complex machine that has its limits. Global illumination cannot be achieved with solutions based on triangle traversal alone. Hardware performance has reached a point that is likely to result in a technological shift in the market in the near future. With beam-based solutions, most of these problems are eliminated immediately, as the different image synthesis there is already basically a realistic picture, no other additional “tricks” are needed.

The other weight that computer graphics carry is the representation of a triangle. For simpler applications, polygon representation is appropriate, but also a major constraint. Polygon-based representation and GPU is the reason why today’s games are "static." The structure of the environment cannot be changed, a wall cannot be destroyed, etc. In some AAA games, similar solutions can be found, but in each case, based on a pre-“wired” logic, the object breaks down and the wall collapses. The reason for the lack of dynamism is the triangular representation. Building a long wall from two triangles and some effects means adequate rasterization speed and quality, but there is not enough information to modify or puncture the wall structure. To create a hole in the wall, the polygon mesh must be modified in real time, on which multiple textures are

usually applied. So based on the location and logic of the fracture, new vertices and triangles must be created, and all of these must be associated with some texture coordinate. This is not an easy task in itself, but since the affected geometry data is stored in the GPU memory due to the fast visualization, all this would have to be brought back to the CPU side into the main memory, made the necessary modifications and then reloaded into the GPU memory. It can be seen that the implementation of this would require extreme complexity and it is not excluded that such tasks cannot be fully solved with this technique. The last problem that arises is to further improve the image quality. Video cards "don't like" a large number of vertices, so developers usually design the models of the game world that saves on polygons where possible. Because curvatures require a higher-resolution polygon mesh, the gaps can be discovered the most at these points.

Although GPUs have evolved a lot over the years, they still try to increase the detail economy of a graphics arena not primarily by increasing the number of vertices, but mainly by increasing surface effects. Such a solution for a brick wall is, for example, Parallax Mapping, Relief Mapping, etc. GPUs have evolved a lot in this area, being able to handle many textures with high performance at the same time. However, texture effects cannot bring image quality even closer to reality in the long run. One of the key factors in improving quality is to increase the resolution of the geometric mesh if the brick wall were constructed as a detailed polygon mesh of many triangles and vertices. Ideally, as the performance of GPUs increases, they will be able to model an increasingly complex world in which objects are made up of more and less triangles. Just think of a forest with a ton of details.

If manufacturers continue this trend, the GPU will have to work with smaller and smaller triangles. Rasterizing tiny details results in very tiny polygons is not a suitable alternative for the GPU (and CPU) either now and in the distant future. This is because during the triangle filling process each triangle involves a so-called setup cost [10]. For a world made up of too many tiny triangles, performance is expected to be consumed by the many setup costs.

GPUs from OpenGL 4.0 and DirectX 11 allow for so-called hardware-accelerated GPU-based tessellation [1]. Manufacturers have also recognized the natural need that arises when the camera gets closer to a geometric object. As long as the object is away or there is no need for a very detailed model, since the small details are not visible. Arriving at the object, however, all the triangles (even more) are needed. A proven solution from previous years was to create multiple detail versions of the same object and then show the version needed (LOD - Level of Detail) as a function of the camera distance.

Tessellation can break down a given surface into additional triangles depending on the distance to the camera. This technique is very useful, it could seem to fix the problems mentioned earlier, but it is a real-time tessellation that can be controlled by the programmer but everything is automated. It is clear that the details of the entire virtual world cannot be left to such a solution. Although it generally improves the end result, in many cases a tessellated object cannot be distinguished from a properly applied Displacement Mapping.

In the (far) future, triangular filling and the polygon representation itself will disappear from computer visualization. When every little detail of an object is needed, you would need tiny triangles, but in this case we reach the level of voxels.

4. Voxel based solutions

Voxel-based visualization is not new in computer visualization, it was available from the beginning, but early slow hardware performance was not yet ready for an approach based on atomic architecture. They also had limited capabilities in memory and storage, so it's no wonder polygon-based image synthesis has become dominant. Initially, the rasterization process was done by only one central unit, only later did the graphics accelerator hardware appear.

Nevertheless, the technology has been constantly evolving over the years, mainly due to computer games, but today it seems to be one of the most important trends in the field of photorealistic visualization. The essence of the name and method is that, in contrast to the polygon-based representation that is common today, it builds a model from so-called voxels (volumetric pixels), which can also be called a voxel set. What a voxel means is difficult to define, and is often referred to in the literature as a three-dimensional pixel, but it can even be called an atom. A typical representation usually includes position, extent, and color information in the model. In addition, rendering engines using different technologies can store other information (such as a normal vector) that is used for a more realistic visualization (such as Ambient Occlusion). They are also used in medical image processing, to display geological data, and of course in (mainly old) computer games (e.g. Blade Runner, Delta Force, Crysis, Voxatron, Minecraft, Motocross Stunt Racer, Red Alert, etc.) to model terrain and various smaller and larger objects. The voxel sets have the advantage that the model can be arbitrarily modified if required, resulting in a destructive environment. However, all of this comes at a high cost, a large data set. When the position and orientation of objects change in space, the voxels have to be transformed one by one. This in itself means a lot of data that requires a lot of memory and CPU / GPU computing power. Although many computer games in the literature used voxel technology, there

were very few (e.g. Hexplore, Voxelstein, Voxatron, etc.) that would have built all of the game world objects entirely on voxels [2]. They often had some limitations (e.g., the number of degrees of freedom). If we want to work with voxels, we run into several obstacles. There is no suitable toolkit at present, most modeling software available on the market can only handle polygon mesh. In addition to static objects, animation is also required. Skeleton-based voxel animation software is not currently available on the market. Nevertheless, we have already encountered this solution in the Voxelstein 3D game.

One of the barriers to the spread of voxel-based technologies is that today's graphics hardware does not directly support the rendering of a voxel set. There are new directions and clever tricks based mainly on ray-based rendering, but there is no suitable uniform direction of support from GPU manufacturers for efficient rendering as we can see in polygon-based solutions.

While it is enough to specify a set of vertices and textures in case of polygon rendering and the GPU is able to render the model directly, for voxel-based models the programmer must create his own shader to implement this (ray-based solutions). The rendering engines of the future are expected to be based on voxel technology when the required computing capacity is available. Nowadays, they can be observed as an additional technique to support for example the calculation of global illumination lights. But there are also efforts to model the entire virtual world in real time (e.g., Atomontage Engine [11], Unlimited Details [12]). Here, however, the limitations of GPUs can already be felt significantly. Although GPUs are fast, the capacity of memory is limited. GPU manufacturers are constantly expanding their technology, up to 6GB cards are now available, but because of the limitation we can't upload any size of a virtual world to the video card memory. In addition, if we need to modify the details (voxels) of the world, we would also need to move the required data to the CPU side. In the absence of this, games will still remain static.

5. Computer visualization of the future

What the technology of the future will bring is hard to predict. However, the trends and the signs of change are already noticeable at the moment. If we look at the future from the point of view of rasterization, we can state that polygon-based rendering will remain with us for a long time to come. However, more and more technology demos are coming from either the ray tracing side or the world of voxels. Voxels, as an atomic-level form of representation, will result in significant improvements in the structure, detail, and modifiability of the playing field. And the dominant display approach in the future is likely to be represented by some ray-based solutions. These will increase the visual quality without applying any of the other tricks and constraints that were

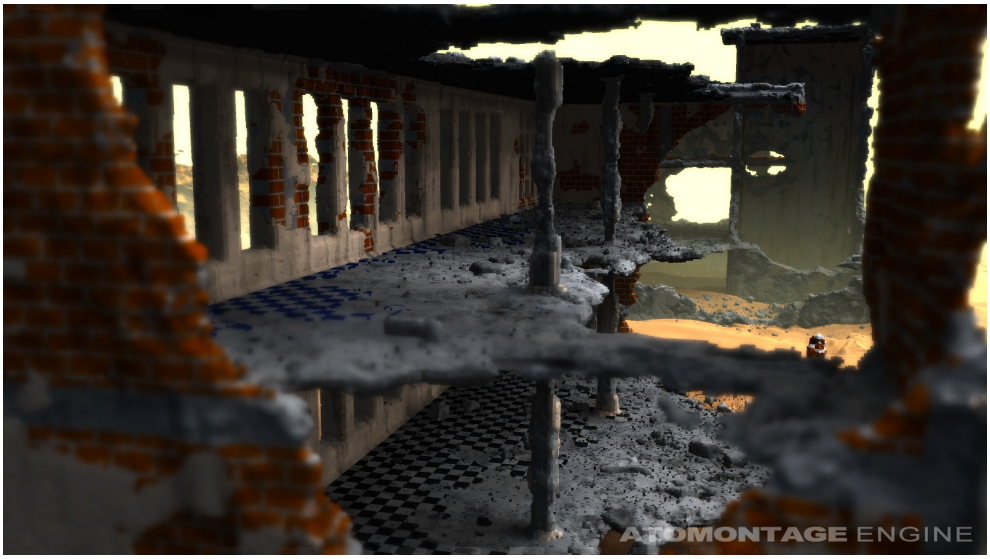

Figure 3. Ruined voxel building with modifiable environment. Excerpt from the Atomontage graphics engine demo

required for a polygon-based approach. The change in technology is unlikely to happen quickly. Modern game engines should also be prepared for voxel-based rendering, and even some hybrid rendering should be used during the transition period. The change in technology is unlikely to happen quickly. Modern game engines should also be prepared for voxel-based rendering, and even some hybrid rendering approach should be used during the transition period. In the history of computer games, there are already some that use a hybrid representation: for example, the game world used a normal polygon-based method, but other various game elements were implemented as voxels. In future solutions, the number of voxel elements and, of course, their quality may increase. Mainly targeting elements that have sufficient richness of detail, as these are the areas that are suitable for voxelation. Transforming a flat surface into a voxel model that does not have any surface characteristics is currently not worth it.

If we approach the rasterization problem from the hardware side, another path can also be outlined. Currently, the market is fully utilizing the GPU for visualization, the price of which has increased significantly in recent times. We have to pay a lot for a good video card today. Previously, the trend was that in exchange for the same amount, we were able to buy a significantly better card in a few years. This trend isn't entirely true today, because a card that runs today's modern games properly costs significantly more than before.

In addition to GPUs, the continuous increase in CPU performance is clearly visible. If we investigate the graphics programming solely from a programming perspective, the CPU would certainly be the right tool for development in the distant future. The reason for this is that the programming model and language of the graphics would not differ from the usual environment, more memory would be available, and data exchange would be easier and more flexible. An experimental video card called Intel Larrabee attempted to accomplish this goal [13]. The architecture was a hybrid approach between the GPU and the CPU, where the programming of the graphics card was based on the x86 instruction set. Modern CPUs with a number of cores (e.g. AMD Ryzen 9 - 12 core, 24 thread) are now available for general use. Thus, a possible direction for the future would be that the central unit should be not only responsible for the business logic but also can take some rendering tasks.The result is a shared computational model similar to that already exemplified in the literature (Battlefield 3, a multi-thread tile-based software rendering technique is outlined where only the CPU is used for light calculations and had great performance results.). This results in a hybrid architecture that combines the benefits of both the CPU and the GPU side. After all, it is not necessarily a good solution if the CPU side is busy with only 2-3 threads and the rest is not used.

6. Conclusion

Today, the field of computer visualization is dominated by polygon-based model representation, and graphics hardware manufacturers have based their rasterization-accelerating hardware on this approach. Although the hardware has been constantly evolving over the years, so far there has been no opportunity to replace the rasterization model based on the classic triangle painting. Nowadays, there seems to be a strong drive on the part of GPU manufacturers to apply real-time ray tracing. This would move the visualization significantly forward, even on a different footing, even if it were to take a step back in image quality from current, highly effect-based solutions. Although the "greatness" of the GPU is constantly pouring out of the marketing machinery, developers will once again realize the potential of the CPU, as 12-core desktops are now available. CPU-based visualizations are expected to provide a more flexible environment whether to display richly detailed voxel sets or a beam-tracking rasterized polygon world.

References

[1] Akenine-möller, T., Haines, E.: *Real-Time Rendering*. A. K. Peters. 3nd Edition, 2008.

[2] Mileff P., Dudra J.: *Simplified Voxel Based Visualization*, Production Systems and Information Engineering, Volume 8, pp. 5-18., 2019. https://doi.org/10.32968/psaie.2019.001

[3] Samuli L., Tero K: *Efficient Sparse Voxel Octrees – Analysis, Extensions, and Implementation*, NVIDIA Technical Report NVR-2010-001, February 2010.

[4] Daniel P: *Tracing Rays Through the Cloud*,Intel Developer Zone, 2012.

[5] Szymon, R., Marc, L.: *QSplat: A Multiresolution Point Rendering System for Large Meshes*. SIGGRAPH '00 Proceedings of the 27th annual conference on Computer graphics and interactive techniques, 2000

[6] Cyril, C., Fabrice, N., Sylvain, L., Elmar, E.: *GigaVoxels: ray-guided streaming for efficient and detailed voxel rendering*. I3D '09 Proceedings of the 2009 symposium on Interactive 3D graphics and games, pp 15-22, 2009. https://doi.org/10.1145/1507149.1507152

[7] Cyril, C., Fabrice, N., Miguel, S., Simon, G., Elmar, E.: *Interactive Indirect Illumination Using Voxel Cone Tracing*. Computer Graphics Forum, Volume 30, Issue 7, pages 1921–1930, September 2011. https://doi.org/10.1111/j.1467-8659.2011.02063.x

[8] Sinje, T., Thorsten, G., Stefan, M.: *Voxel-based global illumination*. I3D '11 Symposium on Interactive 3D Graphics and Games, pp 103-110, 2011. https://doi.org/10.1145/1944745.1944763

[9] Akenine-möller, T., Haines, E.: *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*, Apress, 2019.

[10] Mileff P., Károly, N., Dudra J.: *Accelerated Half-Space Triangle Rasterization*, Acta Polytechnica Hungarica, Volume 12, Issue Number 7, 2015, pp. 217-236. https://doi.org/10.12700/aph.12.7.2015.7.13

[11] Voxel-based Atomontage Engine: *https://www.atomontage.com/*, 2022.

[12] Euclideon Unimited 3D: *https://www.euclideon.com/*, 2022

[13] Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R., Espasa, R., Grochowski, E., Juan, T., Hanrahan, P: *Larrabee: a Many-Core x86 Architecture for Visual Computing*, ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2008 Volume 27 Issue 3, August 2008. https://doi.org/10.1109/hotchips.2008.7476560

[14] Yifan S., Nicolas B. A., Shi D., David K.: *Summarizing CPU and GPU Design Trends with Product Data*,Distributed, Parallel, and Cluster Computing, 2019.

[15] W. J. Dally, S. W. Keckler and D. B. Kirk: *Evolution of the Graphics Processing Unit (GPU)*, IEEE Micro, vol. 41, no. 6, pp. 42-51,2021. https://doi.org/10.1109/mm.2021.3113475