Production Systems and Information Engineering
Volume 8 (2019), pp. 5–18
https://doi.org/10.32968/psaie.2019.001

SIMPLIFIED VOXEL BASED VISUALIZATION

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

Judit Dudra
Bay Zoltán Nonprofit Ltd., Hungary
Department of Structural Integrity
judit.dudra@bay-zoltan.hu

[Accepted December 2017]

Abstract. Voxel-based technology has already been present in the early years of computer visualization. However its usage was almost entirely confined to the field of medical image synthesis. Due to the continuous development of the CPU and graphics hardware, its role has increased again nowadays. It came to the fore especially in the field of computer games because of its specific properties. This publication presents a simplified visualization model and optimization techniques, which are able to visualize smaller (perhaps animated) voxel sets in real-time with certain compromises. The objective of the solution is not to reach the level of photorealistic image synthesis, but a provide a well applicable and simple method which can be applied for computer games and other graphical applications

Keywords: voxel based visualization, software rendering, optimization

1. Introduction

Almost the entire area of the computer visualization is dominated today by GPU rasterized polygon models. Although the voxel-based approach has been available from the very beginning, but the early slow hardware were not yet ready in performance for an approach based on atomic model structure. They were strongly limited in memory and storage, so it is no wonder that the filling based (scanline, half space approaches) polygon image synthesis has become dominant.

In the early ages, the process of rasterization was performed entirely and exclusively by the CPU, the graphics accelerator hardware appeared only later. However, the technology has been continuously evolved over the years, mainly due to the computer games. Today it seems that voxel based technology is a major trend in the area of photorealistic visualization. Besides the growing of hardware performance, the expectation about visualization have also increased. A modern computer game has millions of polygons on the screen at the same time. All of the voxelized versions of these would consume a lot of memory and CPU/GPU time. Today's GPUs are able to render a smaller scene with various real-time effects (e.g. lighting, depth of field, shadows etc), but in cases where many models are on the screen, the available GPU memory is probably insufficient. This led to the unfolding techniques of real-time background data streaming into GPU memory.

To summarize, we can state that modern computer graphics stands before the introduction of the effective voxel technology. Companies, the greatest players in the computer game industry are constantly looking for voxel based complementary solutions to achieve a more realistic view. These techniques, algorithms are complicated, require a high level of knowledge from different areas. This article therefore does not focus on photorealistic visualization, rather displaying smaller voxel sets in a simple way. To answer the question how voxel sets can we visualize without a deep mathematical knowledge beside acceptable quality compromises.

2. Related works

The voxel-based, alternative rendering technology has already appeared in the early years of the computer visualization. Because early hardware did not allow to use displaying models which are able to provide professional, high images quality, thus voxel-based techniques became less known. The main area of its application was the medical diagnostic image synthesis, but some interesting computer games were born applying ray base solutions at the rendering. Although the CPUs were not yet fast enough, but owing to the so-called Wave Surfing algorithm, low-end hardware were able to visualize three dimension models (mainly terrains) in real time. Good examples are: Comanche - 1992, Delta Force - 1998, Armored Fist - 1994, Blade Runner - 1997, Hexplore - 1998 etc. The early consoles (Amiga, Nintendo, Gameboy) used similar technology with the help of software rasterization.

After the continuous spread of the GPU based rendering, which supported the polygon based approach, the software based (voxel) solutions fell more and more into the background. Although there have always been software products (Outcast - 1999, Motocross Stunt Racer - 2002, Red Alert series), which used the voxel approach, the polygon-based technologies come to the fore.

In recent years, voxels have become to a frequented area again. Due to the evolution of the GPU, mainly the approaches using some kind of ray-based technology (ray-tracing, cone tracing etc.) are showing progress. Solutions applying tree structures for faster rasterization are also popular. In publication [3], one of the leader of the GPU market, the NVidia examines the possibilities of using voxel representations as a generic way for expressing complex and feature-rich geometry on current and future GPUs. Their benchmarks show that the octal tree based voxel representation is competitive with triangle-based representations in terms of ray casting performance. Szymon and Marc [4] describe a system for representing and progressively displaying complex meshes that combines a multiresolution hierarchy based on bounding spheres with a point based rendering system. In [11] sparse voxel database structure (GVDB) was investigated based on the voxel database topology. GVDB introduces an indexed memory pooling design for dynamic topology, and a novel hierarchical traversal for efficient raytracing on the GPU. They method can give large performance improvements over CPU methods.

Global illumination is a modern trend to make scene lighting more realistic. The voxel representation is becoming more important in this area. The leading company, Unreal Technologies presented a voxelisation based model for global illumination, applying Deferred rendering [6]. The importance of this technique lies in the fact that the technology is ready for the modern computer games. Paper [7] and [8] also approach the problem of modern lighting using voxel-based technology.

The voxel-based image synthesis is constantly evolving, it will be seen on the screen of every average computer within a few years.

3. Voxel based visualization

Voxel based visualization is not a new area in the field of digital image synthesis. The idea behind the name is that the model builds the 3D model from voxels instead of describing it by today's popular a polygon mesh. It can be difficult to define the term voxel. The literature often calls it as three-dimensional pixel, but can also be called as atom. A typical voxel representation contains the position, size and color information. In addition, renderers using different technologies can store other information (e.g. normal vector), which are used to achieve a more realistic visualization used (e.g. Ambient Occlusion). The voxel-based representation has several advantages over polygon-based storage model. Since the voxel set includes all necessary information of the model for displaying, texture mapping and mip mapping are not needed. The color stored in voxels determines exactly the "look" of the model. Another advantage of the representation is that we can define a very detailed model structure, built upon atomic units. If observe the current trends in the computer game industry, we can declare that the voxel technology can be a very possible future of the computer graphics. This conclusion comes from the fact, that the global objective of real time computer visualization is to reach more realistic, physically correct results on the screen. Therefore the number of applied triangles are constantly increasing and their size is getting smaller. This is a process which tends to an atomic level, which is the world of voxels. Nowadays, screen space solutions (Normal Mapping, Occlusion Mapping, Parallax Mapping etc) are dominated against complex triangle based model structures, because using current screen space algorithms to add detail to a model are cheaper for the GPU, than working with millions of triangles.

The main downside of the voxel technology is hidden in the large data set. Even less detailed model structure has a relatively large set of voxels. Big data set of models requires large amount of main and GPU memory, which is fairly limited in case of GPU based visualization. Various, so-called streaming technology must be developed to keep the current possible visible parts of the virtual world in the GPU memory, however this technology needs a very complex computer and visualization knowledge. Further additional problem of the voxel set is the following: because the voxel based world comprises a large number of voxels, the graphics pipeline should handle large data sets during the various transformations. Because every CPU or GPU cycle counts, this property has a significant impact on performance [1]. Today's graphics hardware does not directly support the voxel based visualization. There are some trends emerged, mainly based on ray-based solutions, but there is no officially supported direction/pipeline from the GPU manufacturers and graphics APIs (OpenGL, DirectX) like in case of polygon-based solutions. using polygons it is enough to specify a set of vertices, textures, and other properties of the object, the GPU can visualize the data almost directly. To visualize a voxel set with the GPU, the programmer usually should develop a custom technology using (sometimes very complex) shaders performing some kind of simplified ray-based technology.

Today, the NVidia company provides a ray tracing engine called Optix, which is able to perform ray tracing on their GPUs. But since it is not officially supported by the common graphics APIs, its application possibilities are limited. The computer game industry will not use this technology until it is not part of the the graphical APIs (OpenGL, DirectX) and uniformly supported on all modern GPU.

The representation of the voxel set is independent of the displaying process. In practice, many volume rendering algorithms have been developed. This paper reviews the most important approaches and also presents a software based simplified solution.

3.1. Cube based approach

This approach is the easiest, we might say the naive approach to display voxel sets. During the visualization, all elements of the voxel set is represented by a three-dimensional cube on the screen. The size of the cube is usually predetermined. Although it is a simple technique, it is fairly popular in today computer games (e.g. Minecraft, FEZ, Stonehearth, Voxatron etc). There are several reasons for this: the visualization model is simple, easy to understand and the results are cubic having retro impressions.

Although the model seems to be simple, because basically every cube is specified by a specific color, in practice, displaying larger models (millions of cubes) cause many serious problems to the GPU due the high number of polygons. As typical example, the shadow creations process can be mentioned. If we use the popular shadow mapping technique, the models need to be renderer in multiple passes. Handling point lights are even more complex, because cubemap is required to store the 6 depth textures in case of only one light. The another approach, shadow volume has also problem with large data sets: this algorithm builds a coherent polygon mesh in real time from the world vertices which are visible from the light perspective. In order to achieve a reasonable frame rate, a number of additional optimization process (e.g. Space partitioning, Occlusion Culling, Ordering objects by z coordinate etc) should be introduced, which makes this technique also complex at the professional level.

3.2. Ray based solutions

An another popular rasterization form of a voxel-based world are the ray-based approaches. A simplified variant (Wave surfing algorithm-2D raycasting) of this technique has already been developed and used in early computer games (Comanche, Wolfeinstein, Delta Force, Armored Fist, Outcast etc) and medical diagnostic procedures. The main idea behind these approaches is that the rasterization problems and the hidden surface determination are solved independently by pixel to pixel. The algorithm casts rays through the render target (e.g. the screen) pixels to the virtual space, then it recursively examines their traversal, collision points and features. The big advantage of the procedure lies in its simplicity. The algorithm can solve numerous visualization problem, which are able to be achieved with current “forward” and “deferred”

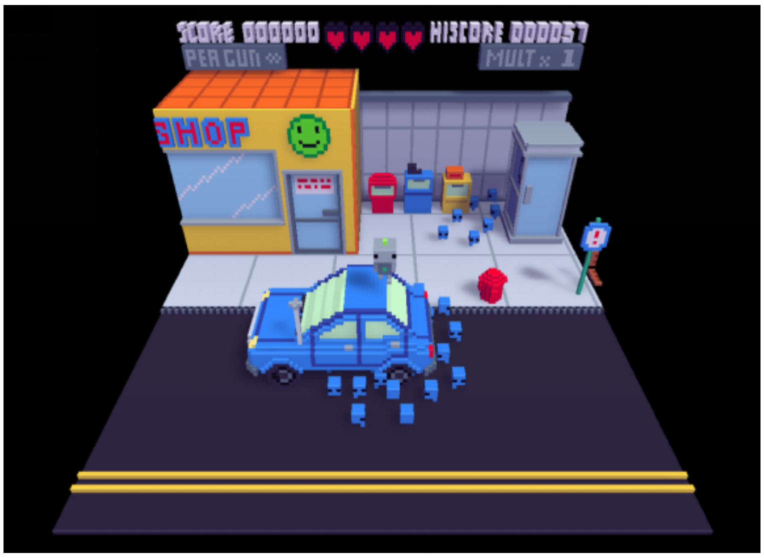

Figure 1. Example of a cube based voxel visualization (Voxatron game)

rendering techniques, but using only a variety of additional techniques that require deep technological and mathematical knowledge (e.g. Cascaded Shadow mapping, Ambient Occlusion etc).

Raytracing and its variants (e.g. path tracing) is the primary starting point for today's global illumination rendering solutions. Today, many attempts unfolds, where the target of the experiment is to develop real time global illumination using voxel based representations. Example: Epic Games-Sparse Voxel Octree Global Illumination, ID Software - Sparse Voxel Octree, NVidia - Efficient Sparse Voxel Octrees [3].

Performing global illumination like ray tracing on a voxel set is a computationally intensive, slow process. Therefore it is essential to use accelerator structures, which are usually based on some kind of space partitioning tree (e.g. KD-tree, BSP tree etc.). Using these structures, the visualization process is the following: the casted rays are hit with the tree levels in order to speed up the voxel searching process, which color should be drawn on the screen. The main disadvantage of ray based solutions is that it is not supported by official graphics APIs(OpenGL, DirectX). Although the graphics pipeline is programmable via shaders, the GPUs are not designed to support the ray-based algorithms by directly by hardware. Today's fast GPUs are able to achieve some ray based rendering in real time. However these algorithms are currently applied only in graphical demos, not in commercial projects (like games).

4. Simplified voxel rendering

The briefly described above techniques are not able to serve all demands. However, there are cases when we want to display smaller voxel sets with acceptable performance and with certain visual compromises (reduced shading/shadows etc.). Good examples are the two-dimensional computer games, where although the projection is two-dimensional, some simple pickable items, models are three-dimensional, they rotate or perform simple animation. Because the display does not want to rent a retro characteristic to the screen, using larger cubes can not be a solution. The rendered voxel set should reflect a two-dimensional characteristic on the screen, but stored in a 3D voxel model format. Figure 2 shows the nature of the approach.

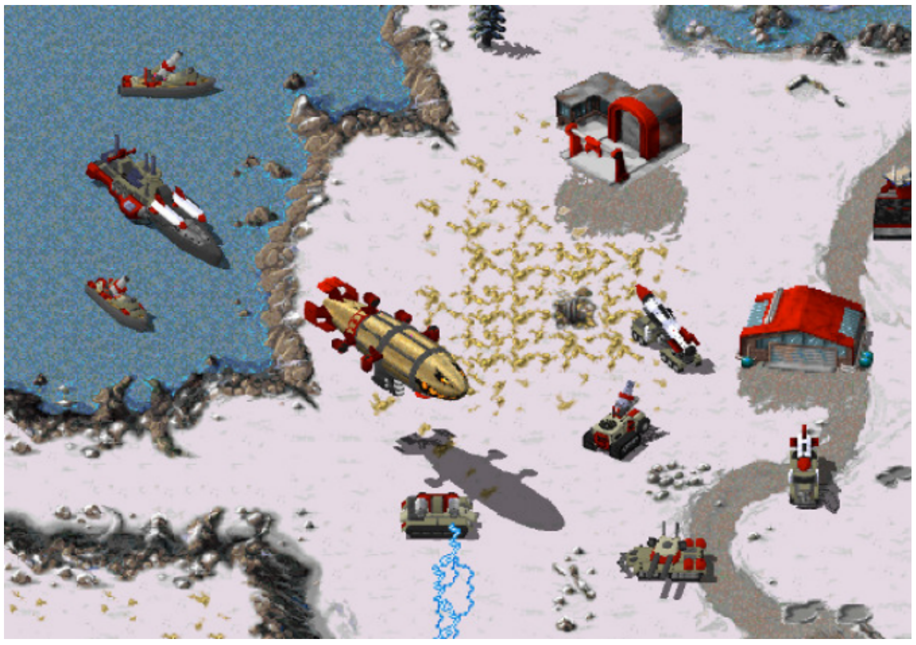

Figure 2. Red Alert, 2D game with 3D voxel units

None of the above solutions are not fully capable of performing these type of rendering tasks. In case of the cube based approach, very small cubes (or possibly balls) could be used in order to achieve high quality game objects. Almost every cube would represent one pixel on the screen. In this case, an unnecessary amount of vertex set is created, which would impose a serious load on the GPU.

The question may arise: why do we need a voxel set to represent a game object, why not implement polygons? In these cases, the polygon set itself would also be dense. But an another important cause is that the developers would like to use the characteristic of the voxel representation, that the object is built from atomic parts and therefore it is destructive.

In the following a simplified voxel rendering solution is presented, which is capable to perform the above-defined tasks efficiently.

4.1. The square based approach

To describe the approach of the simplified voxel renderer, let start from the logic of the displaying process. A voxel is a three-dimensional unit, its rasterisation generally requires a two-dimensional mathematical projection, where the corresponding pixels should be determined in function of the voxel color and size. However the question arises, that if the voxel is mapped to the two-dimensional space anyway, why bother the three-dimensional extent? Can the extent of the voxel be modeled in two dimensions only?

We can apply a simple square based mapping to solve the problem. This technique allows to visualize small voxel sets well in real time under certain compromises. The following figure shows this simple mapping:

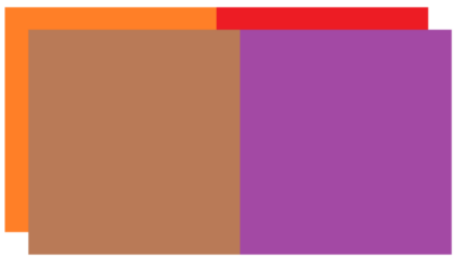

Figure 3. The image shows a magnified mapping of 4 voxels. Two voxels are next to each other and two voxels are located behind

So, during the displaying process, voxels are represented as painted "tiles" (square/rectangle) with predetermined size. The color of the tile is the color determined by the voxel. The difference in x and y directions between the two rows is a natural feature of three-dimensional mapping. In the above example, the first row is located ahead in the space and the model isn't positioned in the origo.

The following pseudo code describes the mathematical model which maps a voxel into the two dimensional space.

```
var per_z = 1.0f / voxel.z;
var square_size = SIZE_FACTOR;

$x_{s1}$ = + ((voxel.y - square_size) * per_z) + half_screen_width;
$y_{s1}$ = - ((voxel.x - square_size) * per_z) + half_screen_height;
$x_{s2}$ = + ((voxel.y + square_size) * per_z) + half_screen_width;
$y_{s2}$ = - ((voxel.x + square_size) * per_z) + half_screen_height;
```

The result can be seen on Figure 4:

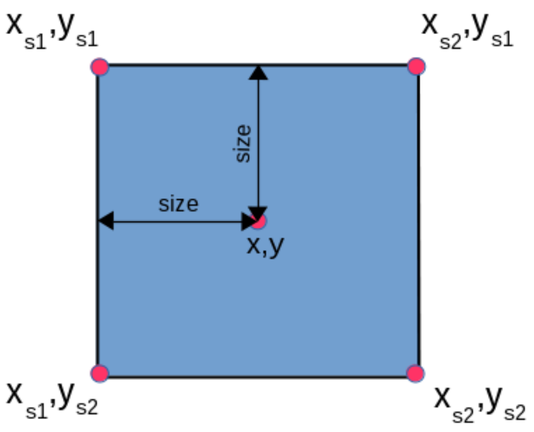

Figure 4. The shape of a screen mapped voxel

The (x, y, z) in the above formulas represent the spatial coordinates of a voxel, the `square_size` is an empirical parameter of the two-dimensional mapping, the size of the square and the x_{s1}, y_{s1}, x_{s2}, y_{s2} coordinates are the corners of the 2D square. `square_size` parameter can be configured according to the rendering requirements. Usually this parameter is constant. Its behavior is the following: if the voxel position is far from the view plane, the model will be smaller on the screen. Therefore we can adjust this parameter to a smaller number, because of the distance of the mapped voxel centers will be also closer and their shape will overlap each other. If the model is closer to the new view plane, the distance between the voxels will be larger. For this reason, the size parameter should be a bigger number, otherwise there will be gaps in the mapped model on the screen as the following image shows:

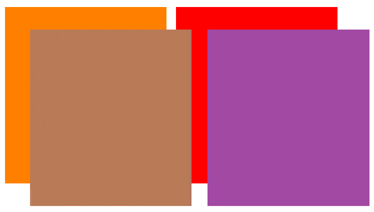

Figure 5. Gap between projected voxels

4.1.1. Rasterizing the voxels

The easiest way to draw a voxel is to use a software based rendering approach. Software rendering uses the power of CPU to create the final voxel graphics. During the rendering process each voxel is drawn into a framebuffer which is located in the main memory, then the buffer is passed to the GPU and will be displayed on the screen. The solution is not incompatible with the GPU-based visualization: the real objective in this case is to integrate the two rendering techniques. During or after the hardware rendered models, the software framebuffer can also be drawn.

The following code describes the drawing of a voxel:

```
for i = x_scr to x_scr2
  for j = y_scr to y_scr2
    index = i * frame_buffer_width + j;
    if (j > frame_buffer_width || i > frame_buffer_height || j < 0 ||
        i < 0)
      continue;
    if (voxel.z < zBuffer[index] )
       z_buffer[index] = voxel.z;
       frameBuffer[index] = voxel.color;
    end
  end
end
```

It is clearly visible, that the solution also requires a z-buffer implementation. The reason for this is that, due to the spatial location of the voxel set, the mapped squares can overlap each other in certain areas. The following image summarizes the result of this technique:

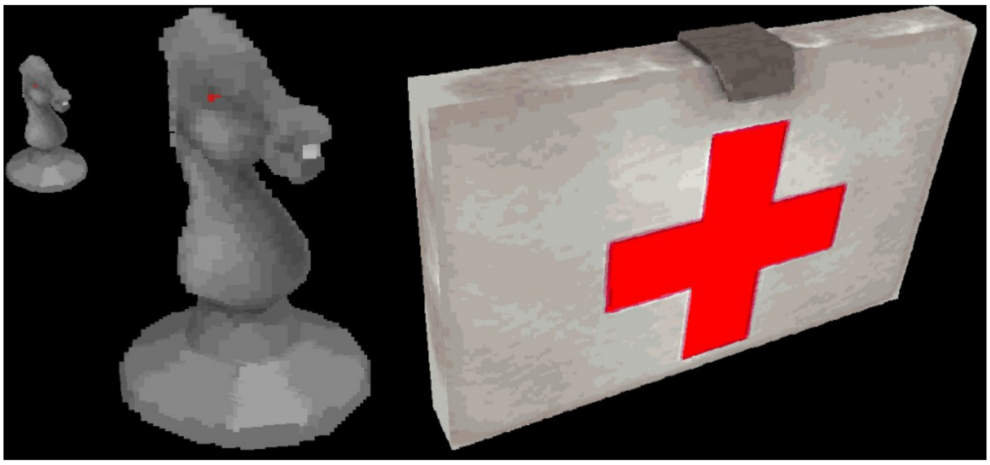

Figure 6. Voxel models in case of different z values

Figure 6 shows two models, where the first model appears twice with different z values. It can be seen that viewing the horse figure from a distance we can obtain satisfactory result, but if from closer the characteristics of the rectangular representation appear. The level of angularity can be eliminated by the tuning of the voxel density and its size. However the rendering time may increase significantly, as the rendering of the second models shows. The medkit model consist of 1.548.288 voxels. Obviously the visual quality is much better, but while 355 FPS was measured at the horse model (49.152 voxels), the performance was dropped to 63 FPS (without optimization) in case of rendering the medkit.

For this reason, this approach is mainly intended for rendering small models in real-time locating not too close to the camera.

5. Accelerating the rendering process

The procedure - although it does not contain any complex mathematical formula - is quite computationally intensive because of the double iteration loop. For every voxel an area of the framebuffer should be colored. The situation is much worse when the voxels are located in the way, that they need to be draw from back to front. In this case, due to the ordering of the voxels, the z buffer cannot reject the non-visible pixels and a lot of pixels will be continuously overwritten. Therefore this rendering technique requires some additional extension, to reduce the number of necessary iterations.

One of the most important optimization is to skip the non-visible voxels of the model. This requires to build a suitable representation for the model, which is capable to determine the outer voxels. During the rasterization the load of the renderer can be reduced significantly.

A further acceleration approach can be the following: if the position of the model changes along the z direction, after a certain z distance it makes no sense to draw squares into the framebuffer. Because of the distance, the boundary points of the mapped voxel are slipped to each other, therefore it is enough to assign a simple pixel to a voxel. With this extension, the performance can be also improved.

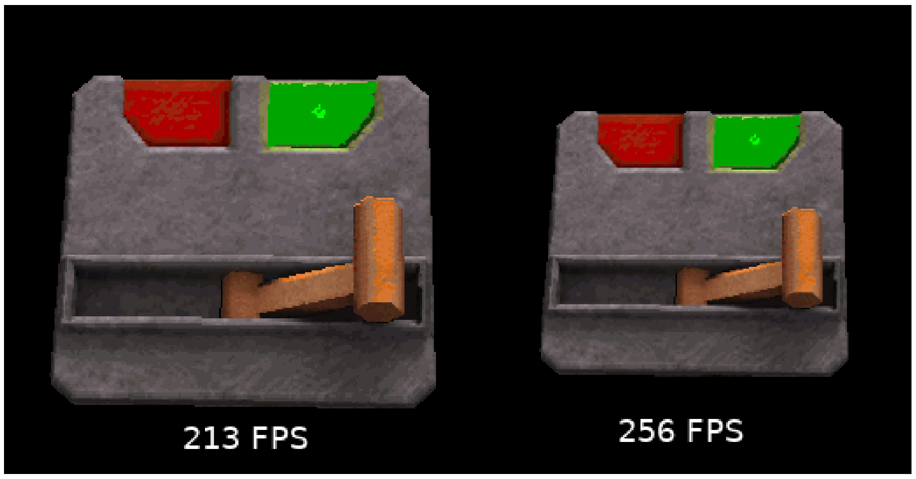

Figure 7. Significant jump in performance: left image is rendered as true squares, right image is rendered as points with different z distance. Number of voxels: 1.548.288

The redundant rasterization due to the formerly mentioned “unfortunately” order of rendering can also be eliminated. Two solutions arise: On the one hand, we can sort voxels along the z direction, then the drawing should began with the nearest. This minimizes the number of overdraws because the z buffer will reject the pixels. The second option is to determine the visible voxels based on the angle of the camera and the model and display only the visible parts.

Implementation of the solution requires substantial changes to the basic structure, where the properties of the voxels are stored separately. An enclosing data structure should be developed that clearly identifies the adjacent voxels. This structure and the approach has the advantage that in this case during the transformations, the calculations shouldn't be performed on each voxel

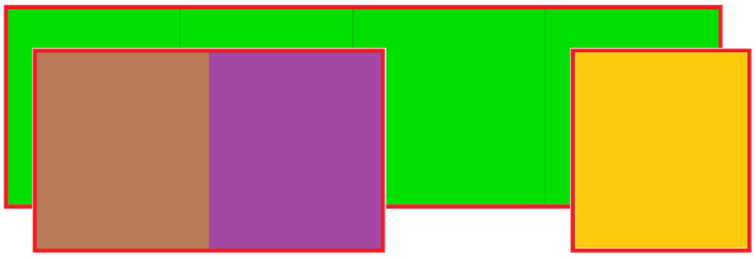

Figure 8. Sample coherent voxel subsets. During the calculations voxels can be handled in one unit, which increases the performance.

separately, but it is enough to be done on the coherent subset (Figure 7.). During the rasterization, the renderer can use these precalculated structure level parameters for drawing and calculating the position of the individual voxels.

6. Conclusion

The field of computer visualization is dominated by the polygon model representations. The graphics hardware vendors based their GPUs on this approach. The voxel technology has been present from the beginning, but it only began to emerge in recent years due to the performance of computers. The method presented in this paper attempts to fill a gap at the field of popular voxel rendering algorithms, which allows fast rendering for smaller voxel sets under certain visual compromises. The solution cannot compete with the performance of the polygon-based visualization, however due to the beneficial properties of voxelization it is well applicable in certain areas. A good example of this is, when the graphics engine combines the voxel-based solutions to the polygon-based visualization opening the way to a modern mixed rendering technology.

Acknowledgements

This research was supported by the European Union and the Hungarian State, co-financed by the European Regional Development Fund in the framework of the GINOP-2.3.4-15-2016-00004 project, aimed to promote the cooperation between the higher education and the industry.

REFERENCES

[1] P. Mileff and J. Dudra: Efficient 2D Software Rendering. Production Systems and Information Engineering, Volume 6, 2012. pp. 99-110.

[2] Akenine-möller T. and Haines, E.: Real-Time Rendering. A. K. Peters. 3nd Edition, 2008. *https://doi.org/10.1201/b10644*

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

[4] Rusinkiewicz S. and Levoy M.: A Multiresolution Point Rendering System for Large Meshes. SIGGRAPH '00 Proceedings of the 27th annual conference on Computer Graphics and Interactive techniques, 2000.

[5] Crassin C., Neyret F., Lefebvre S. and Eisemann 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*

[6] Mittring M. and Haines, E.: The Technology Behind the Unreal Engine 4 Elemental Demo. The 39th International Conference and Exhibition on Computer Graphics and Interactive Techniques, 2012.

[7] Crassin C., Neyret F., Sainz M., Eisemann E. and Green S.: 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] Thiedemann S., Grosch T. and Müller S.: 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] Vega-Higuera F., Hastreiter P., Fahlbusch R. and Greiner G.: High performance volume splatting for visualization of neurovascular data. Visualization, 2005. VIS 05. IEEE, pp 271 - 278, 2005. *https://doi.org/10.1109/vis.2005.47*

[10] Sakamoto, N., Nonaka, J., Koyamada, K. and Tanaka, S.: Particle-based volume rendering. Visualization, APVIS '07. 6th International Asia-Pacific Symposium on, pp 129 - 132, 2007. *https://doi.org/10.1109/apvis.2007.329287*

[11] Rama Karl, H.: GVDB: raytracing sparse voxel database structures on the GPU. Proceeding HPG '16 Proceedings of High Performance Graphics, Pages 109-117, 2016