ISSN 2047-3338

The Hardware Adapted Ray Tracing Algorithm

Yevgeniy Borodavka, Oleksandr Lisovyi and Dmytro Deineka

***Abstract* — This paper is dedicated to solving a problem of the realistic 3D-models rendering with a single FPGA chip. The suggested solution is based on the ray tracing algorithm with pre-processing of the input data by the BVH-tree and with the improved method of the background pixels separation. We regard sphere as the basic 3D-object. Correspondingly, we suggest an improvement method of computing ray/sphere intersection. Algorithm is optimized by reducing the amount of arithmetic operations to the minimum. Therefore, it can be implemented in a single FPGA chip. Furthermore, we suggest a specific method of spheres intersection handling. Software implementation of the suggested algorithm is used for testing and for handling possible errors.**

***Index Terms* — Ray Tracing, FPGA, BVH-Tree, Background Pixels, Ray Casting and 3D-Object**

I. Introduction

The problem of creation of fast and accurate rendering algorithm is very essential. Nowadays, there are a lot of different methods that deal with this problem. However, in most cases, the already suggested methods are intended to accurate rendering of static scenes. The main algorithm for rendering of 3D-objects is the ray tracing (Fig. 1).

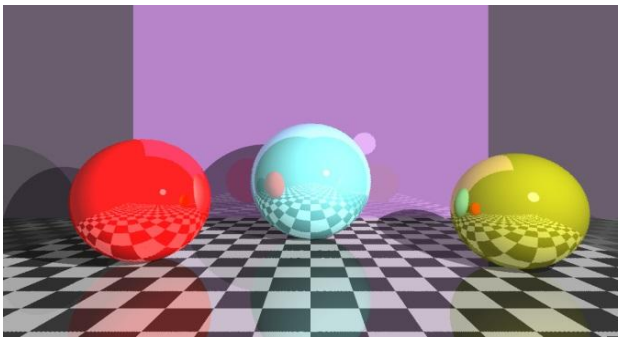

Fig. 1. A scene that is rendered by the ray tracing algorithm

Yevgeniy Borodavka is with Samsung Research and Development Institute Ukraine (SRK), Kyiv 01032, Ukraine, (Email: y.borodavka@samsung.com)

Oleksandr Lisovyi is with Samsung Research and Development Institute Ukraine (SRK), Kyiv 01032, Ukraine, (Email: o.lisovyi@samsung.com)

Dmytro Deineka is with Samsung Research and Development Institute Ukraine (SRK), Kyiv 01032, Ukraine, (Email: d.deineka@samsung.com)

The ray tracing algorithm has received a lot of improvements and adaptations since 1970's. Clear examples are the following: interactive ray tracing for dynamic scenes [1], the hardware architecture for ray tracing [2] and SGRT [15].

The common problem of the ray tracing algorithm is a great number of numerical calculations. The time of scene rendering linearly depends on objects quantity and picture resolution. Developers are forced to downgrade picture resolution or to use less objects in order to reduce calculations.

We do not want to downgrade a picture resolution and we will try to keep quantity of objects as many as possible.

Therefore, we have decided to create the ray tracing algorithm which provides a balance between performance and accuracy. This algorithm must be suitable for implementation in Field-Programmable Gate Array (FPGA) chip.

II. Previous Work

There are several related works which are using hardware implementation of the ray tracing algorithm. The most known works are the following: SaarCOR, RPU, SGRT and HART.

SaarCOR [4] includes a transformation unit for ray transformation. This hardware architecture uses kd-tree acceleration structure but without dynamic updating.

RPU [7] was designed for real-time ray tracing of dynamic scenes with programmable material, geometry, and illumination shaders. RPU uses kd-tree acceleration structure and is implemented in a single FPGA chip. But architecture of RPU supports very low image resolution for real-time processing.

SGRT [15] is mobile ray-tracing hardware architecture for static scenes. It combines dedicated T&I (Traverse and Intersect) units and SRPs (Samsung reconfigurable processors). SGRT provides real-time ray tracing performance at full HD resolution that can compete with that of existing desktop GPU ray tracers. BVH-tree acceleration structure is used.

HART [9] is hybrid architecture for ray tracing. It uses CPU for BVH-tree building and hardware implementation for ray tracing. Axis aligned bounding boxes (AABB) are used as bounding volumes.

All previously mention works are used triangles as primitives whereas our solution is based on spheres.

III. The Algorithm Input Data

Let's suppose that all objects in the dynamic 3D-scene are represented by their bounded spheres. Each sphere has a color (Color), a radius (R), a coefficient of reflection (A), a unique identifier (ID) and coordinates of a center point (C). The screen has HD resolution (1280x720 pixels) and is parallel to the xOy plane at some distance (D). The normal unit vector of the screen is [0 0 1]. The viewpoint of observer (V) is located on Z axis and light position is L (Fig. 2).

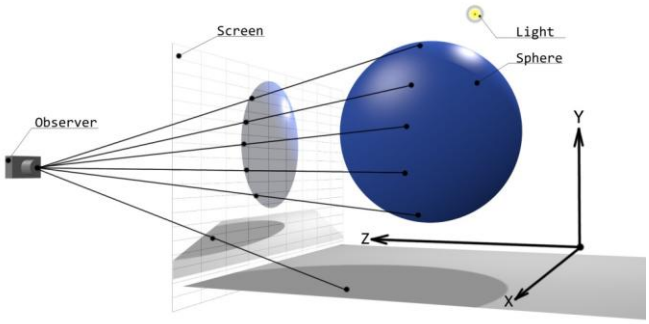

Fig. 2. Illustration to the input data of the algorithm

Our purpose is to generate a realistic image of each object of the scene using the ray tracing algorithm. In direct algorithm we need to cast rays through each pixel of the screen and hence we search for objects that are intersected. We suggest to use Bounding-Volume Hierarchies (BVH) tree in order to reduce a quantity of objects that need to be checked. Likewise, we suggest to separate background pixels in order to reduce a quantity of pixels that need to be traced. Finally, we suggest an improved method for ray/sphere intersection check and a method for handling spheres intersection cases. Let's take a closer look at each of these suggestions in more details.

IV. BVH-tree

We have chosen the BVH-tree as acceleration data structure because it is simple to create and to use. Firstly, we need to sort all out sphere in correct direction. Secondly, we need to arrange spheres by 4 and create high levels of the tree.

A. Sphere Sorting

The viewpoint is placed at Z axis in positive direction. All objects are placed behind xOy plane in negative direction of Z axis. Thereby, we need to sort spheres from near to far in relation to observer. We use only z-coordinate of the spheres center for sorting, i.e. the radius is not taken into consideration. It implies that the front side of some sphere with a larger radius may be the nearest to screen even if its center point has a bigger distance than another sphere with smaller radius (Fig. 3).

This kind of spheres' sorting helps us to create BVH-tree rapidly and optimally.

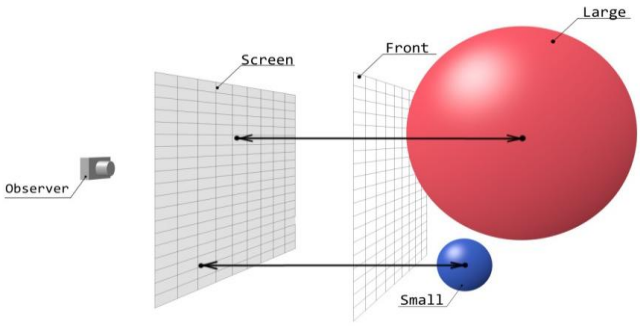

Fig. 3. Example of spheres sorting

B. Tree Construction

When all spheres are arranged, we can start to construct the BVH-tree. Every node in the tree is the sphere. Every level of the tree has no more than 4 children and every leaf node is placed at the lowest level. This structure gives us acceleration to locate object sphere with several comparisons because at every tree level we select only one node. The algorithm of the tree creation is described below.

The strategy of the tree creation is a merge of the spheres by 4 in the order of approach to the screen. During the merging process we also compute a spatial box ($P_{Near}(X_N, Y_N, Z_N)$, $P_{Far}(X_F, Y_F, Z_F)$) that includes all 4 spheres. Also, we compute the maximum spatial box that includes all spheres ($P_{Min}(X_{Min}, Y_{Min}, Z_{Min})$, $P_{Max}(X_{Max}, Y_{Max}, Z_{Max})$). After the input spheres are merged, the next level of spheres is created. The merging process is repeated to each sphere at the current level.

The center point (C) and the radius (R) of a new sphere is computed by the following equations:

$$X_C = \frac{X_N + X_F}{2}; Y_C = \frac{Y_N + Y_F}{2}; Z_C = \frac{Z_N + Z_F}{2};$$

$$R = \sqrt{(X_N - X_C)^2 + (Y_N - Y_C)^2 + (Z_N - Z_C)^2}$$

If radius (R) of the new sphere is greater than the radius that was computed for maximal spatial box, then new sphere has replaced by the sphere that was constructed on the maximal spatial box. We have implemented this rule to reduce incremental increasing of the new spheres radius.

For N spheres the BVH-tree has a maximum depth of $[(\log_2 N)/2] + 1$. At each level the nodes are placed from left to right in order of increasing distance from the screen (Fig. 4).

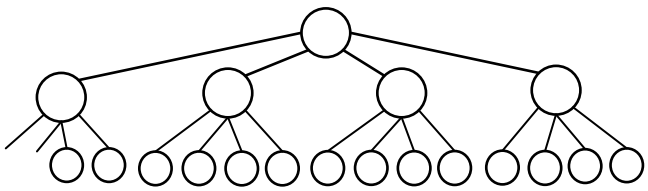

Fig. 4. BVH-tree example

The traversing of the tree with ray is very simple. We move from the root to leaves and from left to right at each level. If ray intersects the node's sphere, then we move one level down. When ray intersection with leaf sphere is detected, the traversal stops. In best case, we will revise not more than $[(\log_2 N)/2] + 4$ spheres, in the worst case it will be $4*[(\log_2 N)/2] + 1$ spheres.

V. Background Pixels Separation

When the BVH-tree construction is complete, then the stage of background pixels separation starts. The 3D-objects do not usually fill the whole scene. Thereby we do not have to trace pixels from the background. We suggest detecting background pixels with the following two methods. The difference between these two methods is the strategy of the basic points' determination.

A. Projection Matrix

Both suggested methods are used the projection matrix to create projection of the root sphere basic points on the screen:

$$\begin{bmatrix} Z_V + D & 0 & 0 & 0 \\ 0 & Z_V + D & 0 & 0 \\ -X_V & -Y_V & D & -1 \\ -X_V \cdot D & -Y_V \cdot D & -Z_V \cdot D & Z_V \end{bmatrix}$$

This projection matrix depends of the viewpoint (V) position and the distance from the screen to xOy plane (D). Any spatial point P(X, Y, Z) will be projected to the screen point p(x, y, z) with the following coordinates:

$$x = \frac{X \cdot (Z_V + D) - X_V \cdot (Z + D)}{Z_V - Z}$$

$$y = \frac{Y \cdot (Z_V + D) - Y_V \cdot (Z + D)}{Z_V - Z}$$

$$z = -D$$

B. First Accurate Method

First method is very accurate but it needs many computations. The strategy of the basic points' determination is the computation of at least 64 points on the sphere and projecting the points on the screen. That means that we need to use 16 meridians and 16 parallels to determine the basic points. More meridians and parallels can be used for more accurate projection. The basic point P(X, Y, Z) can be computed by the following equations:

$$X = X_C + R \cdot \sin(\alpha) \cdot \cos(\beta)$$

$$Y = Y_C + R \cdot \sin(\alpha) \cdot \sin(\beta)$$

$$Z = Z_C + R \cdot \cos(\alpha)$$

$$\alpha \in [0; \pi], \beta \in [0; 2\pi]$$

This method is useful when the viewpoint can be shifted in any directions from its origin. In this case we have a perspective deformation of the scene's objects. Big amount of the basic points helps us to make more accurate rendering of the objects.

We have several points of the sphere on the screen when projection is complete (Fig. 5).

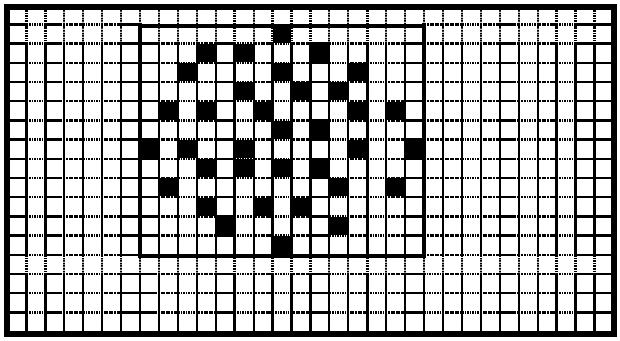

Fig. 5. Sample of a sphere projection on the screen

Firstly, we need to compute a minimal bounding box (MBB) that includes all pixels of the projection. Then we need to restore a shape of the sphere at the screen projection. We scan every line in MBB to detect first and last pixel of the projection. All pixels between first and last one in each row we need to mark as the projected pixels (Fig. 6).

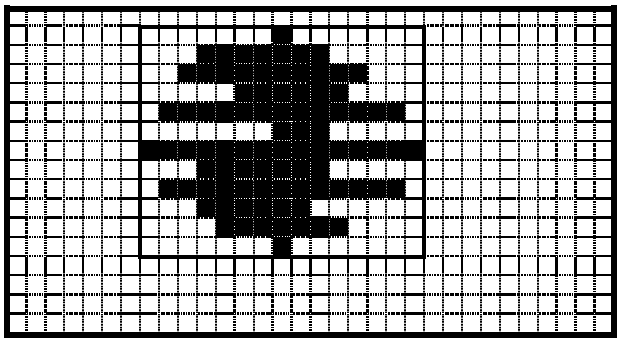

Fig. 6. Pixels of the projection after horizontal scan

The next step is vertical scan for filling the omitted pixels. Last step is marking border's pixels as projected to prevent possible errors (Fig. 7).

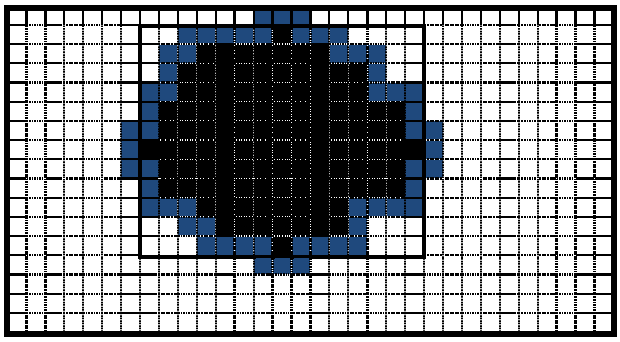

Fig. 7. Final projection of the root sphere on the screen

Each pixel of the screen that is not marked is the background pixel. We can use for these pixels the background color of the scene. Thus, these pixels need not to be traced by the rays.

C. Second Fast Method

Coordinates of points at sphere computation is a very disadvantageous process with many arithmetical operations. We suggest to use only 4 basis points for projection to reduce computations. These points we compute by the following equations:

$$X_{P1} = X_C + 1.4; Y_{P1} = Y_{C;}$$
$$X_{P2} = X_C - 1.4; Y_{P2} = Y_{C;}$$
$$Y_{P3} = Y_C + 1.4; X_{P3} = X_{C;}$$
$$Y_{P4} = Y_C - 1.4; X_{P4} = X_{C;}$$

We use coefficient 1.4 to prevent errors in object's projections when direction of view is less than 90° to the screen (Fig. 8).

As we follow this method we have square screen projection with edge 2.8R. In this case we have 2.5 times more background pixels marked as projected pixels than in the first method:

$$\frac{(2.8)^2 \cdot R^2}{\pi \cdot R^2} \approx 2.5$$

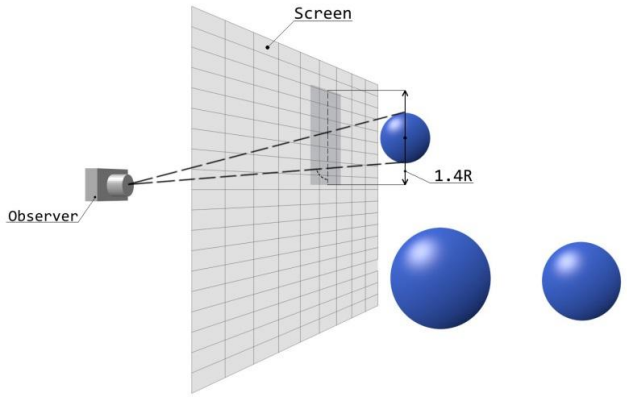

Fig. 8. Illustration of using the coefficient

However, we still separate most of the background pixels and reduce a number of pixels for tracing. Hence, we use only 2 additions and 2 subtractions for all 4 points as compared to 3 additions, 5 multiply and 4 trigonometrically functions for each point in the first method.

After comparison of two methods we have made a conclusion that the second method is less disadvantageous and we use it in our hardware implementation.

VI. Ray Casting

Now, we have the tree and the rectangular area of pixels that have to be traced. For each pixel (P) we need to create a ray (VP) with origin at the viewpoint (V). We traverse the tree with this ray from the root to the leaves and from left to right at each level until the ray/sphere intersection is found. The main problem in this case is the computation of the point of ray/sphere intersection. However, we need to compute the intersection point only if it is a leaf sphere. The only thing we need to know for node spheres is whether there is intersection or not.

When intersected sphere is found and we have computed the intersection point, we make a reflected ray and combine current object color with a color of the reflected object. After that we also cast a ray into a light direction to compute the final color of an object.

A. Ray/Sphere Intersection Point

The geometrical method presented in [3] is used to find intersection point. This method is illustrated at Fig. 9.

The distance between the sphere center and the ray computed by Pythagorean equation is:

$$D^2 = L^2 - T^2.$$

The squared distance between the viewpoint and the sphere center (L) can be found by the following equation:

$$L^2 = (X_V - X_C)^2 + (Y_V - Y_C)^2 + (Z_V - Z_C)^2.$$

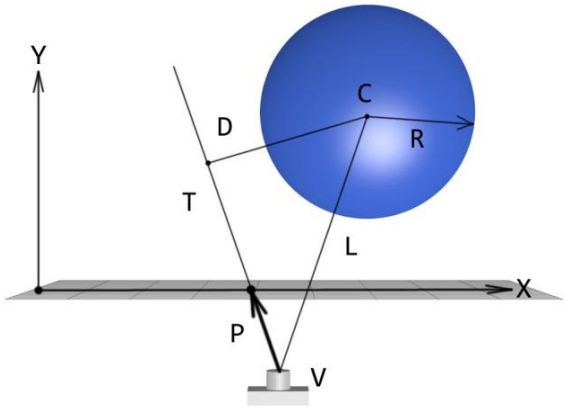

Fig. 9. Ray/sphere intersection illustration

A ray intersects a sphere if the following inequality holds:

$$R^2 - D^2 > 0.$$

The distance from the viewpoint to the point on the ray that is the closest to the sphere center (T):

$$T = \overrightarrow{VC} \cdot \frac{\overrightarrow{VP}}{\left|\overrightarrow{VP}\right|}.$$

After substitution we get the formula:

$$R^2 - L^2 + \left(\overrightarrow{VC} \cdot \frac{\overrightarrow{VP}}{\left| \overrightarrow{VP} \right|} \right)^2 > 0.$$

Note that, if (VC*VP<0) and (R^2<|VC|^2) then there is no intersection at all.

After multiplying each member by |VP|^2 we have:

$$\left(R^2 - L^2\right) \cdot \left| \overrightarrow{VP} \right|^2 + \left(\overrightarrow{VC} \cdot \overrightarrow{VP} \right)^2 > 0.$$

This formula requires minimum number of operations (11 multiplications and 11 additions). This fact is very important for hardware implementation.

In order to compute the intersection point coordinates, parameter T_1 must be calculated first:

$$T_1 = \left(R^2 - L^2\right) + \frac{\left(\overrightarrow{VC} \cdot \overrightarrow{VP} \right)^2}{\left| \overrightarrow{VP} \right|^2}.$$

Now parameter *t* can be computed with the following formula:

$$t = T - \sqrt{T_1}.$$

Hence the intersection point is:

$$\begin{bmatrix} X & Y & Z \end{bmatrix} = \begin{bmatrix} X_V & Y_V & Z_V \end{bmatrix} + t \cdot \begin{bmatrix} X_{VPnorm} & Y_{VPnorm} & Z_{VPnorm} \end{bmatrix}$$

B. Reflected Ray Computing

For more accurate and realistic scene rendering we must use at least one reflection for each object. We assume, that surface of the spheres is absolutely smooth. Hence, we use the following equation for computing the reflected ray:

$$\overrightarrow{MN} = \overrightarrow{VP} - 2 \cdot \left(\overrightarrow{CM} \cdot \overrightarrow{VP} \right) \cdot \overrightarrow{CM},$$

CM stands for the normal vector of the sphere at the intersection point.

Now, we can find intersection of spheres and the reflected ray with a help of algorithm that is described in paragraph 5.A. The only difference in tree traverse is that we need to check all nodes at every level for intersection. At the end of the tree traverse we will be able to have several spheres that are intersected by the ray. Between these spheres we select the nearest one. The color of reflected sphere is used for blending with a color of the found sphere by following equation:

$$Col = (1 - A) \cdot C_F + A \cdot C_R$$

C_F stands for the color of the found sphere, C_R stands for the color of the reflected sphere.

C. Light Source Check

Light source makes a great contribution to object color formation. If light source is occluded by some object, then we will have shadow effect. Otherwise, we need to compute a coefficient of light source contribution to the object's color.

We use the same algorithm that is described above in order to find out whether we have some sphere at light source direction. If there is no sphere at light source direction, then we compute coefficient of the light contribution as triple dot product between normal unit vector of the sphere and unit vector direction to the light position. An example of rendered scene with one light source and two-time reflection is presented at Fig. 10.

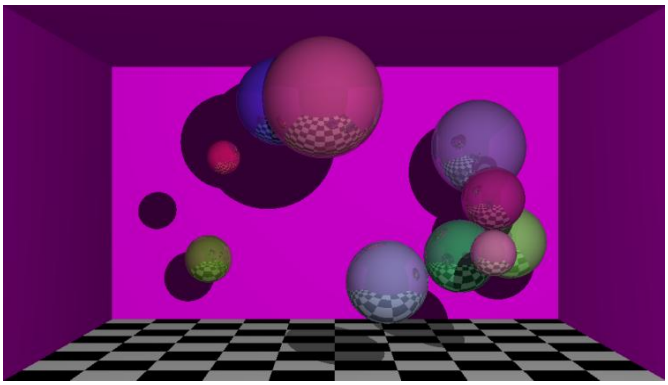

Fig. 10. Example of rendered scene with light and reflection

D. Spheres Intersections Handling

We have a little bug in case of spheres intersections due to simple method of BVH-tree construction (Fig. 11).

As we can see at Fig. 11, the small sphere is found first due to tree construction algorithm, because it is closer to the screen than the bigger one. However, this case is totally incorrect. We suggest simple solution to solve this problem. We make the reflected ray as mentioned in paragraph 5.B and when we find an intersected sphere then we test it. The additional test is very simple and it is as follows: if the origin of the reflected ray is located inside the sphere, then we will find that it is the case of spheres intersection. We use the following formula to check situation when the origin of the ray within the sphere:

$$R^2 - \left| MC \right|^2 > 0,$$

MC stands for the vector from the intersection point (at the small sphere at Fig. 11) to the center point of the reflected sphere (the big one at Fig. 11).

The result of the suggested method of handling intersection cases is presented at Fig. 12. There we have 100 spheres, one light source and two-times reflections.

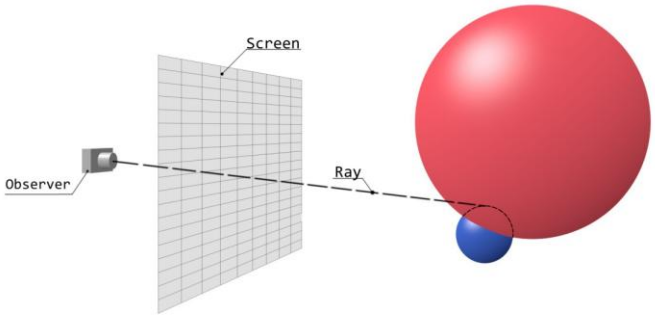

Fig. 11. Case of the spheres intersection

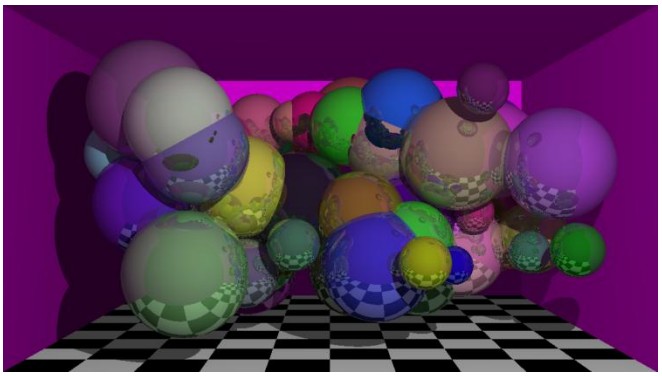

Fig. 12. The result of the suggested method

As you can see at Fig. 12, all cases of the spheres intersections are handled correctly.

VII. Hardware Architecture Concept

Likewise HART [9], our hardware architecture is used external memory and CPU for preprocessing and storing data (Fig. 13).

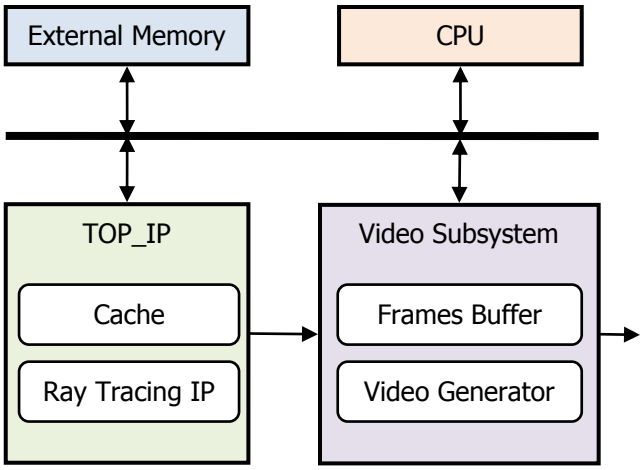

Fig. 13. The top level architecture

The BVH-tree, scene parameters and primitive's parameters are stored in the External Memory.

CPU is used for the following: preparing and saving scene parameters to the External Memory; preparing and saving BVH-tree and primitives parameters to the External Memory; making the background pixels separation; TOP_IP control; Video Generator initialization.

Video Subsystem is used for storing frames in buffer and to prepare video for transmission through HDMI port. Frames buffer is an internal FIFO memory for storing the last 3 frames. It is used for synchronization between the TOP_IP and the Video Subsystem and makes possible output of streaming video. Video Generator generates video signals according to the HDMI protocol. CPU initializes Video Generator with HD video format parameters.

TOP_IP contains Ray Tracing component and Cache. Cache is used to hold the most recently accessed primitives' parameters. Ray Tracing IP generates frames in HD (1280×720 pixel) format. Each pixel is coded by the 24-bit true color format.

Ray Tracing IP architecture is divided into three parts: Rendering, L1 Reflection, and L2 Reflection. In addition, there is a MUX frames component which is used for selection of frames with needed depth of 3D effect. All parts work simultaneously and have access to the External Memory. Each part generates frames with defined depth of 3D effect.

VIII. Conclusions

This work provides the rendering method that was used in patent application [16]. The suggested ray tracing algorithm is designed for hardware implementation and has 3 improvements compared with other algorithms. They are: pre-processing input data with sphere BVH-tree, background pixel separation and very simple formula for ray/sphere intersection detection.

The suggested method can be used in interactive systems with 3D graphic (video games for computers and mobile devices, electronic mapping, Computer Aided Design application etc.).

Today we have software implementation of suggested algorithm and draft hardware implementation with only 5 spheres. Thus, we could not compare our current results with existing state-of-the-art solution, because no one has used sphere as a primitive.

Our future work is to create new hardware architecture on single FPGA chip and to implement there our algorithm.

References

[1] C. Lauterbach, S.-E. Yoon, D. Tuft, and D. Manocha, "RT-DEFORM: Interactive ray tracing of dynamic scenes using BVHs," in *Proceedings of the* 2006 IEEE Symposium on Interactive Ray Tracing, pp. 39-45, 2007.

[2] J. Schmittler, S. Woop, D. Wagner, W.J. Paul, and P. Slusallek, "Real-time ray tracing of dynamic scenes on an FPGA chip," in *Proceedings of the ACM*

SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, pp. 95-106, 2004.

[3] A.S. Glassner, *An Introduction to Ray Tracing*. Academic Press Limited, 1989.

[4] J. Schmittler, I. Wald, and P. Slusallek, "SaarCOR – A Hardware Architecture for Ray Tracing," in *Proceedings of the Graphics Hardware*, pp. 1-11, 2002.

[5] C. Wachter, and A. Keller, "Instant Ray Tracing: The Bounding Interval Hierarchy," in *Proceedings of the 17th* Eurographics Symposium on Rendering, pp. 139-149, 2006.

[6] E. Reinhard, B. Smits, and C. Hansen, "Dynamic Acceleration Structures for Interactive Ray Tracing," in *Proceedings of the* Eurographics Workshop on Rendering Technics, pp. 299-306, 2000.

[7] S. Woop, J. Schmittler, and P. Slusallek, "RPU: A Programmable Ray Tracing Unit for Realtime Ray Tracing," *ACM Transaction on Graphics vol. 24*, pp. 434-444, 2005.

[8] T.J. Purcell, I. Buck, W. R. Mark, and P. Hanraham, "Ray Tracing on Programmable Graphics Hardware," in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 703-712, 2002.

[9] J.H. Nah, J. W. Kim, J. Park, W. J. Lee, J. S. Park, S. Y. Jung, W. C. Park, D. Manocha, and T. D. Han, "HART: A Hybrid Architecture for ray Tracing Animated Scenes," *IEEE Transaction on Visualization and Computer Graphics, Volume 21, Issue 3*, pp. 389-401, 2015.

[10] S.-E. Yoon, S. Curtis, and D. Manocha, "Ray tracing dynamic scenes using selective restructuring," in *Proceedings of Eurographics symposium on rendering 2007*, pp. 73-84, 2007.

[11] J. Bigler, A. Stephens, and S. G. Parker, "Design for parallel interactive ray tracing systems," in *Proceedings of IEEE Symposium on Interactive Ray Tracing 2006*, pp. 187-196, 2006.

[12] Y. Gu, Y. He, K. Fatahalian, and G. Blelloch, "Efficient BVH construction via approximate agglomerative clustering," in *Proceedings of the 5th High-Performance Graphics Conference*, pp. 81-88, 2013.

[13] T. Karras and T. Aila, "Fast parallel construction of high-quality bounding volume hierarchies," in *Proceedings of the 5th High-Performance Graphics Conference*, pp. 89-99, 2013.

[14] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "OptiX: a general purpose ray tracing engine," *ACM Transactions on Graphics (SIGGRAPH 2010)*, vol. 29, no. 4, pp. 66:1–66:13, 2010.

[15] W.-J. Lee, Y. Shin, J. Lee, J.-W. Kim, J.-H. Nah, S.-Y. Jung, S.-H. Lee, H.-S. Park, and T.-D. Han, "SGRT: A mobile GPU architecture for real-time ray tracing," in *Proceedings of the 5th High-Performance Graphics Conference*, pp. 109–119, 2013.

[16] I. Borodavka, O. Lisovyi, and D. Deineka, "Rendering System and Rendering method thereof," US Patent Application 20160005210 July 2014 [Online]. Available: http://www.google.com/ patents/US20160005210

Mr. Yevgeniy Borodavka

Received B.Sc. and M.Sc. degrees in computer science and Ph.D. degree in computer aided design from Kyiv National University of Construction and Architecture, Ukraine, in 2002, 2003 and 2009, respectively. Since 2005 assistant professor and since 2009 associate professor of the Information Technologies of Design and Applied Mathematics department in Kyiv National University of Construction and Architecture, Ukraine. From 2004 to 2009 – lead engineer and from 2009 to 2012 – senior scientific staff in State Enterprise "State Research and Development Institute of Computer Aided Design in Construction". Since April 2013 is lead engineer in Samsung Research and Development Institute Ukraine (SRK).

Mr. Oleksandr Lisovyi

Received B.Sc. and M.Sc. degrees in computer engineering and Ph.D. degree in computer systems and components from V.M. Glushkov Institute of cybernetics of National Academy of Sciences, Ukraine, in 2005, 2006 and 2010, respectively. From 2005 to 2006 – engineer, from 2006 to 2010 – junior research assistant, since 2010 is research assistant in V.M. Glushkov Institute of cybernetics of NAS of Ukraine. From 2011 to 2012 – engineer, from 2012 to 2015 – lead engineer, since 2015 is senior engineer in Samsung Research and Development Institute Ukraine (SRK).

Mr. Dmytro Deineka

Received B.Sc. and M.Sc degrees in electronic engineering from National Technical University of Ukraine "Kyiv Polytechnic Institute" in 2009 and 2011, respectively. From 2010 to 2012 RTL engineer in Design bureau. From 2012 to March 2015 is engineer and since March 2015 is lead engineer in Samsung Research and Development Institute Ukraine (SRK).