Computer science

Oleksandr ROMANIUK,
Doctor of Technical Sciences, Professor,
Vinnytsia National technical university
ORCID ID: 0000-0002-2245-3364

RAY TRACING METHODS FOR FORMING THREE-DIMENSIONAL GRAPHIC IMAGES

Ray tracing[1] rendering techniques are the foundation of photorealistic visualization of 3D scenes because they accurately model the propagation of light, including direct and indirect illumination, reflections, refraction, shadows, and caustics.

Classic ray tracing[2] (Whitted-style Ray Tracing) is a basic method of photorealistic rendering (Fig. 1), proposed by Thierry Whitted in 1980, which laid the foundation for modern global illumination algorithms. In this approach, a primary ray is emitted for each pixel of the image from the observation point (camera) towards the scene. When this ray intersects the surface of the object, the system determines the point of contact and calculates local illumination based on lighting models such as the Phong or Lambert model.

To account for more complex visual effects, new rays are created at the intersection point: reflection, refraction and shadow ray. They are passed according to the optical properties of the material – specularity, transparency, refractive index – and allow the creation of shadows, specular reflections and refraction effects.

The method is deterministic: each ray generates a limited number of new rays, so the computational complexity increases logarithmically with the depth of the reflections. However, because this approach does not account for multiple

1 Romaniuk, ON, Bobko, OL, & Romaniuk, OV (2024, September 26–27). Prospects for using change tracking technology to create photorealistic images in real time. Computer games and multimedia as an innovative approach to communication – 2024: Proceedings of the IV All-Ukrainian Scientific and Technical Conference of Young Scientists, Postgraduates and Students (pp. 327–328). Odesa, Ukraine; Romaniuk, ON, & Zaválnyuk, Ye. K. (2024, May 29–31). Methods of reverse tracing of changes. Development of modern science and education: realities, quality problems, innovations: Proceedings of the V International Scientific-Practical Internet Conference (pp. 119–124). Zaporizhzhia, Ukraine

2 Zeng, J., Bao, C., Chen, R., Dong, Z., Zhang, G., Bao, H., & Cui, Z. (2023). Mirror-NeRF: Learning neural radiance fields for mirrors with Whitted-style ray tracing. ACM Transactions on Graphics, 42(1), 1–13.

scattering of light between surfaces, it does not model global illumination effects such as color casts, caustics, or soft shadows from indirect sources. Despite this, classical ray tracing produces a clear, high-contrast image with sharp shadows, clear reflections, and basic optical effects, making it an important foundational step in the development of more complex tracing systems.

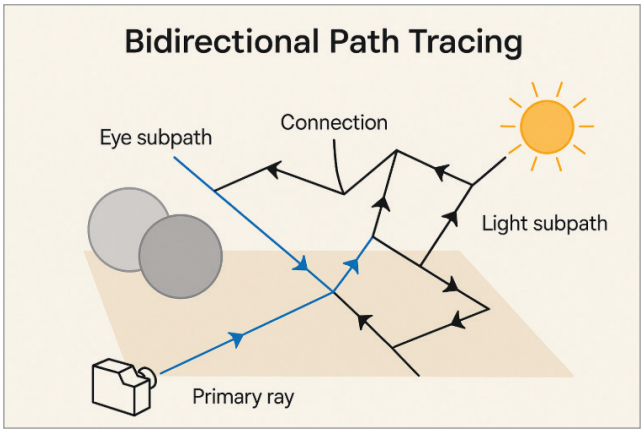

Fig. 1. Whitted-style Ray Tracing

Acceleration systems such as Bounding Volume Hierarchies or kd-trees are commonly used to efficiently find ray intersections with scene geometry, reducing the computational burden. This method is well suited for scenes with solid objects, glass, or mirror materials, where direct rays of light play a major role, rather than multiple reflections or scattering. In graphics engines, classical ray tracing is often combined with other techniques or used as a basis for extensions to path tracing or photon mapping.

Path Tracing[3] is a physically correct global illumination method that models the propagation of light in a 3D environment through stochastic ray tracing. Its basic idea is that for each pixel in the image, a ray is emitted from the camera, which intersects the scene, finds a point of contact with an object, and generates random rays that simulate the reflection of light according to the BRDF function of the surface. New rays can also be generated at each new intersection, until either the ray reaches the source light, or will not be stopped. As a result of this process, the average value of the light energy arriving at the camera along random trajectories is calculated for each pixel.

3 Jakob, W., & Marschner, S. (2010). Path tracing in real-time graphics. Journal of Computer Graphics Techniques (JCGT), 1(2), 34-50. URL: https://www.jcgt.org/published/01/02/02.

Path tracing[4] (Fig. 2) automatically takes into account all components of global illumination: indirect illumination, diffuse overillumination, color translucency, soft shadows, caustics, reflections and refraction. The method is much simpler to implement than, for example, photon mapping or bidirectional path tracing, but requires a large number of samples per pixel to achieve a low noise level. Because of this, an image obtained with a small number of rays looks grainy, and for a high-quality result, either a long calculation or the use of denoising methods is required.

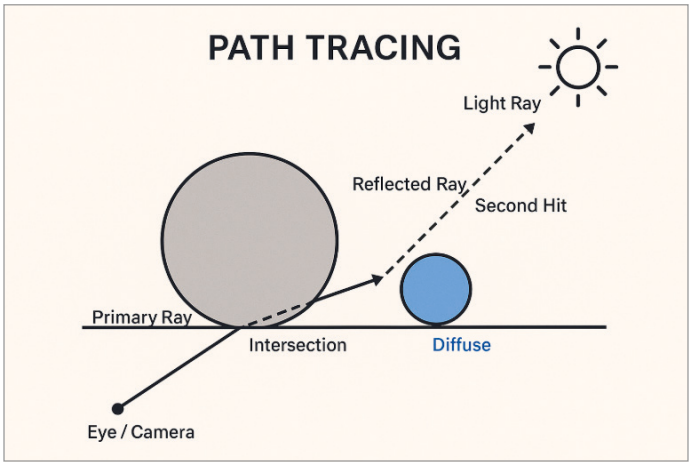

Fig. 2. Path Tracing method

The illumination calculation is done using the Monte Carlo integration method, where the choice of the scattering direction is based on a weighted sampling. An important part of the algorithm is the selection of the direction of the new ray taking into account the BRDF of the material: for mirror surfaces – the direction of reflection, for transparent – refraction, for diffuse – a random direction in the hemisphere. To prevent infinite generation of rays, the path termination technique is used .

Path tracing is widely used in offline rendering, for example, in photorealistic animations, architectural visualization and VFX. Due to its simplicity of implementation and high physical fidelity, it is implemented in renderers such as Cycles (Blender), Arnold (Autodesk), LuxCoreRender, Corona, Redshift (hybrid) and others. In recent years, with the development of GPU computing power and accelerated ray tracing

4 Pharr, M., & Humphreys, G. (2016). Physically based rendering: From theory to implementation (3rd ed.). Morgan Kaufmann. ISBN: 978-0128006450.

technologies (RTX, DXR), the path tracing method has begun to be used in real time, in particular in game engines such as Unreal Engine 5.

In general, path tracing provides the highest level of realism among rendering methods, but it requires significant computational resources. Optimization of sampling, use of denoisers (e.g., based on neural networks), adaptive sampling, and combination with other methods (e.g., with directional lighting or simplified photon mapping) allow its effective application even in dynamic scenes.

Bidirectional Path Tracing[5] (Fig. 3) is an advanced stochastic global illumination method that models light transmission in complex scenes, combining the advantages of conventional path tracing and source-dependent light tracing.

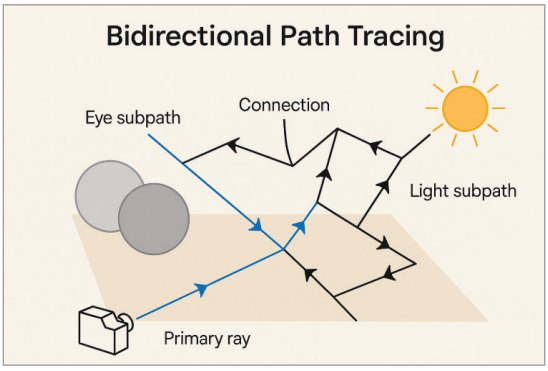

Fig. 3 . Bidirectional Path Tracing

The basic idea of the method is to simultaneously construct two types of paths: one path is created from the camera and the other from the light source. At the points where these two paths intersect or connect, a complete trajectory is created, which used to estimate the contribution of light to a given pixel. Unlike classic path tracing, where light sources can only be reached randomly, bidirectional path tracing allows you to effectively capture all significant light paths, including caustics, indirect lighting, and complex interactions between scene objects.

When constructing each trajectory, the method uses stochastic sampling based on BRDF and light models, similar to standard path tracing. However, additional source-dependent sampling allows for higher efficiency in situations where light is difficult to reach from the camera due to multiple

5 Parker, SG, & Shirley, P. (2008). Bidirectional path tracing in theory and practice. Journal of Computer Graphics Techniques (JCGT), 2(4), 60–85. URL: https://www.jcgt.org/published/02/04/03.

obstacles or high refractive index. Each path consists of a sequence of vertices (nodes), and the algorithm attempts to connect points on the path from the camera to points on the path from the light source to form useful trajectories. For each connected trajectory, a weight is calculated based on the scattering function (BSDF), the scene geometry, and the visibility coefficient between the points.

Since possible connections between subpaths are generated at each step, it is necessary to balance the probability weights of these connections to avoid double counting or noise. To do this, the Multiple Importance Sampling (MIS) technique is used, which optimally combines the results from different path selection strategies, minimizing variance and ensuring stable convergence of the illumination integral. Using MIS allows you to avoid the dominance of less efficient sampling options, for example, when only one of the directions (from the source or from the camera) leads to a significant contribution.

The advantages of bidirectional path tracing are its better ability to simulate complex lighting phenomena, including caustics (e.g., light reflecting off a glass of water), indirect light in enclosed environments, and more stable convergence with a small number of samples per pixel. The main disadvantages are its complexity in implementation, the need for additional data structures for subpath tracing, and the increased computational burden due to the need to connect many combinations of nodes.

This method is implemented in leading offline renderers such as LuxRender, Mitsuba, Arnold and partially in Cycles (Blender) when using lighting sampling. Combined with other techniques – for example, with Metropolis Light Transport (MLT), which use the ideas of path selection optimization – bidirectional path tracing forms the basis of modern physically accurate rendering. Its application is especially effective in scientific visualization, architectural rendering and VFX, where the accuracy of lighting modeling is critically important.

Metropolis Light Transport (MLT) (Fig. 4) is a powerful stochastic rendering algorithm based on the Monte Carlo integration method using stochastic sampling according to the Metropolis method. Its main goal is to efficiently simulate global illumination in complex scenes where classical methods such as path tracing or bidirectional path tracing exhibit slow

convergence or high dispersion. MLT[6] works especially well in complex lighting conditions such as caustics, indirect light, narrow light passages, or scenes with small openings through which light rarely penetrates when randomly sampled.

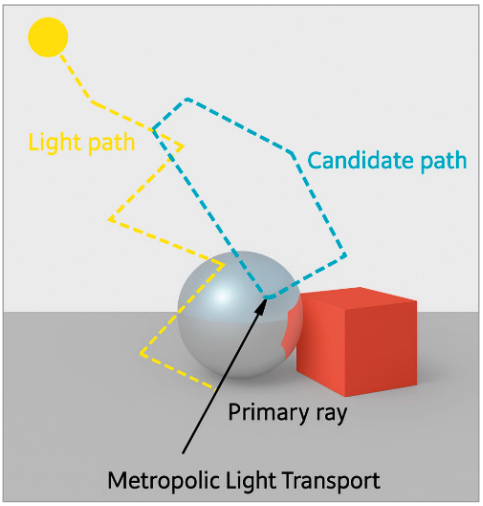

Fig. 4. Metropolis Light Transport method

Unlike classical methods, which generate an independent ray path for each iteration, MLT builds new paths by stochastic modification of an already found path.

If a new trajectory is accepted, it is added to the accumulation of light contribution, and it becomes the basis for further perturbations. If the path is rejected, the previous path is re-considered. Thus, the process simulates a Markov chain, which over time concentrates on important trajectories that have a large contribution to the image. As a result, this allows you to achieve a higher quality result with fewer samples compared to classic Monte Carlo rendering.

One of the key features of MLT is its ability to effectively handle phenomena that are rarely encountered in standard random sampling – for example, light passing through a small slit and focally reflecting off a certain surface.

Along with the main version of MLT, several variations have been developed, including Bidirectional Metropolis Light Transport, which combines the advantages of bidirectional ray tracing with a stochastic trajectory perturbation

6 Keller, A. (1997). *Radiosity and the Metropolis light transport method . Computer Graphics Forum*, 16(2), 99–106. DOI: https://doi.org/10.1111/1467-8659.00123

mechanism, and Primary Sample Space Metropolis Light Transport, which performs perturbations not in the geometric space of trajectories, but in the space of primary sampling parameters, which facilitates implementation and allows the method to be integrated into standard render engines.

Metropolis Light Transport is widely used in high-fidelity offline rendering for VFX, architectural visualization, scientific simulation, and the film industry. Well-known implementations include renderers such as Mitsuba, PBRT, and experimental systems in academic research.

Photon Mapping[7] (Fig. 5.) is a two-step stochastic global illumination method developed by Henrik Wann Jensen in 1995 that allows for efficient modeling of complex lighting effects, including caustics, indirect lighting, and color casts. Unlike classical path tracing, which calculates illumination by generating rays from the camera, photon mapping works by emitting photons from light sources into the scene and then using information about their distribution to calculate illumination at the intersection points of the rays emitted from the camera.

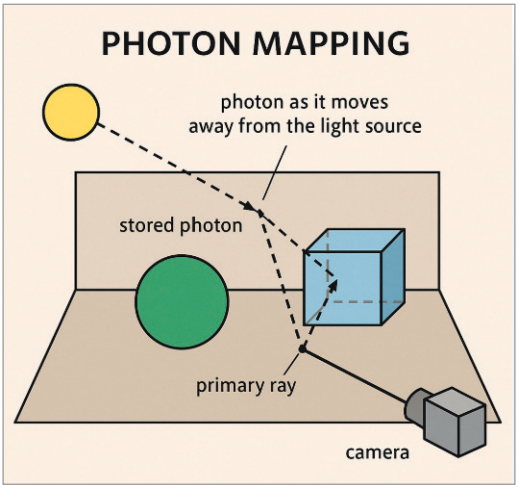

Fig. 5. Photon Mapping

The algorithm consists of two main stages: building a photon map and rendering. In the first stage, light sources emit a large number of photons that spread across the scene, interact with objects, and at certain points of collision, the photons are stored in a special structure – a photon map.

7 Jensen, HW (2001). Photon mapping in real-time rendering . In Proceedings of the 28th annual conference on computer graphics and interactive techniques (pp. 69–76). ACM. DOI: https://doi.org/10.1145/383259.383287.

Each photon has coordinates, direction, energy (or power), and type of interaction with the surface. Typically, separate maps are constructed for caustics (photons that have passed through focusing materials), global illumination, and direct light .

In the second stage, when the camera emits rays for rendering, at the point of their intersection with the scene object, the system performs a local search in the photon map. Within a radius of a certain distance, a set of photons that fell on this area is determined, and based on their intensity and direction, the density of the light flux is calculated. Most often, the density estimation method is used for this, where the total energy of photons in the radius is normalized to the area of the search sphere. This allows you to estimate indirect lighting, taking into account scattering and interaction of light with other objects.

An important advantage of photon mapping is its ability to accurately model caustics, the focused rays that occur, for example, when light passes through glass or water objects. While conventional methods often cannot generate such paths due to their rarity, photon mapping preserves photons in the areas where these effects occur, providing a detailed result with a relatively small number of samples from the camera. This method also provides well-controlled computation through parameters of search radius, number of photons, weighting (e.g. cone filtering) and can be combined with other approaches, such as final gather, to improve quality.

The disadvantages of photon mapping are the need for a complex data structure – usually a kd-tree or priority search tree to quickly find the nearest photons, the complexity of parameter tuning, and the appearance of artifacts at the boundaries of dense and sparse regions of the photon distribution. However, by dividing the computational process into two stages, the method scales very well and can be effectively used for scenes with a large number of light sources and complex geometry.

Photon mapping is actively used in scientific rendering, architectural visualization, simulation of optical effects, and software where control over computing resources is required.

The Metropolis Light Transport (MLT) (Fig.6) method is one of the most effective global illumination techniques used in physically correct rendering of photorealistic images. Unlike classical ray tracing, where a large number of light

paths are generated independently of each other, MLT[8] is based on the idea of mutating already found effective light paths. The algorithm starts (Fig. 6) with the initial path that runs from the camera to the light source, and then makes local changes to this path, creating new variants. Each new path is accepted or rejected based on its contribution to the final image – if the new path gives a significant increase in illumination, the probability of its acceptance increases. This approach allows you to focus on those parts of the illumination space that make the greatest contribution to the final image, which significantly increases the sampling efficiency. The method works in complex scenes with complex geometry, caustics, specular or transparent objects, and small light sources, where traditional tracing or photon mapping methods require a large number of samples to achieve acceptable quality.

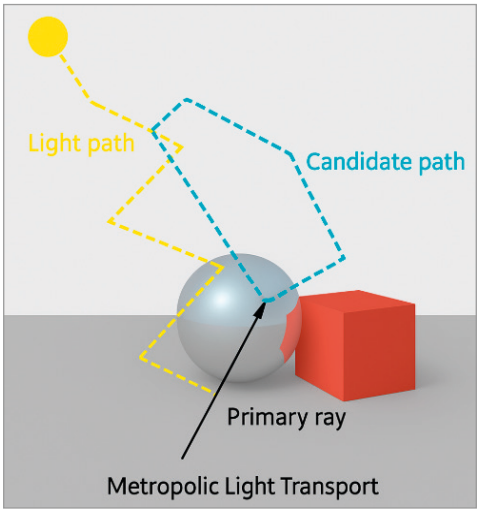

Fig. 6. Metropolis Light Transport method

Real -time[9] ray tracing (Fig. 7) ais a rendering method used in computer graphics to simulate the behavior of light in a scene with a high level of realism.

Unlike traditional rasterization, which approximates lighting effects, ray tracing simulates the physical interactions of light rays as they pass through a scene. This includes reflections, refraction, shadows, and global illumination such as diffuse reflections and caustics.

8 Schmidt, M., Lobachev, O., & Guthe, M. (2016). Coherent Metropolis Light Transport on the GPU using Speculative Mutations. *Journal of WSCG* , 24 (1), 1–8. Retrieved.

9 Schmidt, M., Lobachev, O., & Guthe, M. (2016). Coherent Metropolis Light Transport on the GPU using Speculative Mutations. *Journal of WSCG* , 24 (1), 1–8. Retrieved.

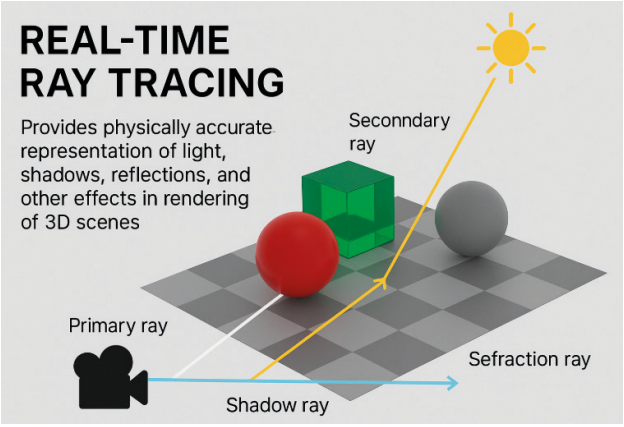

Fig. 7. Real-time ray tracing

To achieve real-time execution on modern graphics hardware, especially on GPUs with ray tracing support (such as the NVIDIA RTX series), these calculations are accelerated using specialized cores, known as RT cores, which are optimized for the parallel processing required for ray tracing operations. To optimize the balance between image quality and performance, denoising and hybrid rendering techniques are often used, where ray tracing is combined with rasterization.

While real-time ray tracing is still computationally expensive, advances in hardware and software, such as using ray tracing only for specific effects (such as reflections and shadows), have made the method more accessible for real-time rendering, resulting in more realistic graphics and immersion in virtual environments.

Reservoir-based Spatiotemporal Importance Resampling (ReSTIR) is a technique used in rendering to improve the efficiency of real-time rendering. Its main goal is to reduce noise in images while maintaining a high level of quality and detail, especially when using ray tracing techniques. The technique works on the principles of importance resampling , which allows you to selectively store the most significant light samples to improve rendering quality.

ReSTIR is based on the use of reservoir resampling , which allows data points (e.g. pixel luminance) to be selected for resampling based on their significance, which can reduce computational cost and noise in the final image. This is important because ray tracing methods can be very computationally demanding, especially in real time.

ReSTIR[10] combines techniques commonly used in traditional rendering with methods that allow for significant resampling in space and time. This allows for significant improvements in lighting accuracy in complex scenes, such as those involving reflections, refraction, or soft lighting, where accuracy and detail are critical to image quality. An important characteristic of ReSTIR is its ability to achieve high image quality without significantly increasing computational costs, thanks to efficient use of memory resources and resampling of the most important points. This allows the use of real-time ray tracing methods, which was previously not possible due to high computational complexity. Due to its ability to improve visual quality while significantly reducing noise and computational costs, ReSTIR is a powerful tool for use in industries such as computer graphics, video games, as well as in real-world visualization systems for simulations and media. This method opens up new possibilities for rendering complex scenes with high efficiency.

Ray Marching[11] (or Sphere Tracing) is a method of rendering implicit surfaces that differs from classical ray tracing in that it does not calculate exact intersections of analytically given geometry (e.g. polygons), but steps along a ray in the direction of the camera using surface distance functions. This method is most often used for rendering scenes (Fig. 8) built on the basis of Signed Distance Functions (SDF) – functions that for any point in space return the minimum distance to the nearest surface: a negative value indicates that the point is inside the object, a positive value indicates that it is outside, and zero indicates that it is on the surface.

The essence of the method is that a ray is emitted from each point on the screen and moves in the direction of the scene. At each step of the movement, the value of the signed distance function is calculated at the current point of the ray – this value is used as a safe step length forward, that is, it is guaranteed not to collide with any surface if you do not approach closer than this distance. Thus, the ray gradually "approaches" the surfaces of objects. If at a certain step the SDF value becomes less than a certain threshold ε (for example, 0.001), it is considered that the ray has reached the surface. In this case, the normal to

10 Noh, H., Lee, H., & Kim, J. (2020). Reservoir-based spatiotemporal importance resampling for efficient environmental monitoring. *Journal of Environmental Informatics* , 32(4), 45-57. URL: https://doi.org/10.1016/j.envint.2020.105021.

11 11 Lengyel, E. (2016). *Ray tracing and ray marching techniques for high-quality graphics*. In Real-Time Rendering (pp. 120-135). CRC Press.

the surface, illumination, reflection and other parameters for the fragment are calculated. If the ray has not reached the surface to the maximum radius or number of steps, it is considered that it has not collided with any object, and the fragment is background.

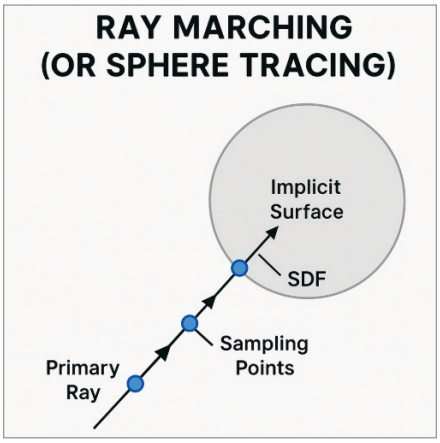

Fig. 8. Ray Marching

Ray marching is particularly well suited for rendering complex fractal objects (e.g. Mandelbulb, Mandelbox), surfaces that cannot be easily described polygonally, or scenes where the geometry is procedurally generated. SDF also allows for easy merging, cutting, or intersecting of objects via Boolean operations on functions (Constructive Solid Geometry – CSG), smooth union, and even real-time deformation of geometry by modifying distance functions.

The advantages of the method include: ease of implementation in fragment shaders (GLSL, HLSL), high flexibility in geometry construction, good scalability for GPU, no need to store complex meshes or BVH structures. Among the disadvantages: the method does not guarantee fast convergence in scenes with sharp surface differences, it is difficult to model complex textures or transparent materials, it is difficult to integrate with classic rendering scenes, and there is a risk of «skipping» through narrow elements of the scene if the SDF incorrectly describes the geometry.

Ray marching is often used in demo scenes, generative art, interactive web projects on ShaderToy, and as a tool in scientific visualization and wave process simulation. Modern applications also include neural fields, where a signed distance function is approximated using small neural networks for 3D reconstruction or content generation.

The implementation of ray tracing methods on GPUs relies on parallel computing and the use of specialized hardware blocks such as RT Cores (in NVIDIA RTX), Ray Accelerators (in AMD RDNA2/3) or Ray Tracing Units (Intel Xe). Below is an overview of how different ray tracing methods are implemented on GPUs:

Ray tracing methods are implemented on GPUs because of their massively parallel nature. Each pixel of the screen or voxel of the scene is processed independently, making the GPU ideal for firing large numbers of rays simultaneously. The simplest implementation on GPUs is classic Whitted-style ray tracing, where a single primary ray is fired for each pixel, and secondary rays (reflections, refraction, shadows) are generated when intersecting with an object. This approach does not take into account indirect lighting, but can be implemented using compute shaders, CUDA, OpenCL, or APIs such as DirectX Raytracing (DXR) and Vulkan RT.

Path tracing, which is a physically correct method of global illumination, is also implemented on the GPU, but with some limitations. The GPU processes thousands of rays in parallel and performs BRDF calculations, scattering direction selection, clipping through BVH, and generating new rays on each bounce. It also uses the Russian Roulette technique to eliminate unimportant paths, and AI denoisers (such as NVIDIA's OptiX) to clean up the result with a small number of samples. Path tracing is supported in engines like Blender Cycles, Redshift, Arnold GPU, and in some interactive demos that use RTX.

Bidirectional path tracing on the GPU is more difficult to implement, as it requires constructing trajectories not only from the camera, but also from light sources. For each such subpath, the GPU stores the collision points and tries to connect them, calculating visibility. This requires more memory, buffers, and complex logic, which limits real-time efficiency. Typically, such an implementation is used in offline renderers (Mitsuba, LuxCoreRender), where speed is not critical.

The Metropolis Light Transport (MLT) method is difficult to implement on GPUs due to its sequential nature. It is based on constructing a single ray path and successively mutating that path. Since such modifications depend on the previous state, full parallelism is difficult. However, batch implementations are possible, where the GPU processes several independent Markov chains

simultaneously. In such implementations, it is important to preserve the current state of the path and effectively apply the Metropolis criterion.

Photon mapping is implemented on the GPU in two stages. In the first stage, the GPU emits millions of photons from light sources that pass through the scene and store them in a photon map – usually in the form of a kd-tree or a spatially indexed buffer. In the second stage, rays from the camera are traced, which at the intersection points look for the nearest photons in this buffer. This allows the GPU to calculate indirect illumination based on the photon density. For efficiency, cone filters, progressive smoothing, or a final gather phase are often used.

Real-time ray tracing is implemented through hardware acceleration. NVIDIA uses RT Cores, which perform ray intersection with triangles and BVH tracing in hardware. DXR in DirectX 12 and Vulkan RT in Vulkan provide the programmer with a new type of shaders – RayGen, ClosestHit, Miss, which the GPU processes with support for BVH structures. A hybrid rendering model, where rasterization performs the basic geometry, and ray tracing only shadows, reflections or global illumination, allows you to achieve a realistic result in real time.

Ray marching is implemented as a pure shader approach on the GPU, usually in fragment shaders (GLSL or HLSL). Each pixel fires a ray that gradually steps along the direction of the scene, using the value of a signed distance function (SDF) to determine a safe step. If the ray reaches the surface, the normal and illumination are calculated, if not, the processing stops. This approach is very effective in procedural rendering, such as in ShaderToy or WebGL, where all the geometry is given by formulas.

The ReSTIR method is implemented on the GPU through a combination of compute shaders that control the selection of light sources, and buffers for temporal and spatial redistribution. At each pixel, the GPU creates a set of lighting candidates, selects them through importance sampling, and stores them in a reservoir (reservoir sampling). Information from previous frames is used to improve stability (temporal reuse), and resampling of neighboring pixels is used for spatial smoothness (spatial reuse). ReSTIR allows tracing only 1–2 rays per pixel, making it suitable for real-time use even in scenes with thousands of light sources.

Therefore, the implementation of ray tracing methods on GPUs largely depends on the type of method: some of them are well parallelized (path tracing,

photon mapping), some are difficult to implement in real time due to recursion (bidirectional, MLT), and the latest methods (RTX, ReSTIR) combine algorithmic optimizations with hardware support, which allows achieving realistic lighting even in dynamic game scenes.

Combining the direct rendering method (rasterization) with the ray tracing method is widely used in modern computer graphics systems, especially in real-time. This approach is called hybrid rendering, and its essence lies in the fact that the basic image of the scene – geometry, textures, materials – is formed by the traditional rasterization method, and the calculation of complex lighting effects, such as soft shadows, specular or blurred reflections, refraction, caustics, global illumination or ambient occlusion – is performed by ray tracing. This allows you to combine the advantages of both approaches: the speed of rasterization and the physical accuracy of tracing.

In hybrid systems, rasterization is used to render the bulk of the scene, including depth buffer calculations, screen pixel filling, and pre-calculation of materials and normals. Ray tracing is then applied only to selected pixels or regions where accurate light simulation is required. For example, in the case of a mirrored surface, a ray is traced from the pixel in the direction of reflection, the intersection with other objects is determined, and the result is blended to the final color. Similarly, for shadows, only one ray can be traced to the light source, checking for obstructions, instead of using traditional shadow maps.

This combination allows for a significant reduction in computational costs compared to full ray tracing. Many engines (e.g. Unreal Engine 5, Unity HDRP, Frostbite) implement hybrid algorithms, where geometry processing and most shading operations are performed by GPU rasterization, and tracing is called only at the post-process level or in individual shaders. Using the DirectX Raytracing (DXR) or Vulkan Ray Tracing API, developers can selectively use TraceRay() in the middle of pixel or compute shaders, while the main part of the scene is rendered by the classic pipeline.

Hybrid rendering also allows for optimized memory usage: for example, trace results can be stored in buffers and reused in subsequent frames or during animation. This opens up possibilities for implementing techniques such as screen-space reflections with fallback to ray tracing or reflections from previous frames.

Thus, the combination of direct rendering and ray tracing is practical and strategically important for achieving photorealism while maintaining performance. This is especially important in game graphics, virtual reality, augmented reality, interactive visualization of architectural scenes, where it is necessary to instantly respond to user or lighting changes. With the development of hardware, in particular RT cores, such a combined scheme has become the standard for modern graphics engines and will be the basis of next-generation rendering.

In addition to the technical feasibility of combining the direct method (rasterization) with ray tracing, there is also a deep conceptual feasibility of such an integration. Direct methods are real-time classics based on the projection of a polygonal scene onto the screen with subsequent light processing via shaders, but they have a number of fundamental limitations. For example, shadows are often created using shadow maps, which have limited resolution and cause artifacts at the edges; reflections are built using environment maps or screen-space reflections, which do not cover objects outside the visible space. The raster pipeline cannot accurately account for multiple light scattering, caustics, or refraction through transparent materials with complex geometry.

Ray tracing, on the other hand, is based on the physical principles of light propagation, but is computationally expensive. This is why the combined approach allows for intelligently distributing computations. Visually simple areas of the scene (e.g., large walls, floors, uniform lighting) are processed by rasterization, while complex areas—transparent objects, shiny surfaces, soft shadows, caustics—are traced selectively. This can be implemented as a dynamic solution in the engine itself: for example, the system automatically determines where it is appropriate to activate ray tracing based on the complexity of the shading or the current camera position.

Modern hybrid renderers implement even deeper levels of integration. For example, ray tracing results are used not only for the final image, but also for building the G-buffer in the deferred rendering pipeline. Rays can also be fired in compute shaders to pre-calculate lighting (for example, in deferred global illumination passes), the results of which are cached in a lighting grid or voxel representation. This opens the way to new methods such as voxel cone tracing, which is itself a hybrid of tracing and volumetric lighting.

Another interesting perspective is the interaction of ray tracing with screen-space methods: for example, screen-space ambient occlusion (SSAO) can be augmented with ray-traced ambient occlusion (RTAO) only in areas where SSAO fails. Similarly, screen-space reflections (SSR) can be replaced with ray traced reflections only in pixels where SSR does not see the necessary geometry (e.g., off-screen or behind objects).

In a hybrid approach, multi-level tracing is also possible – where the first bounce is performed by ray tracing, and subsequent bounces are estimated by empirical models or by approximation using cache or screen-space information. Such compromise schemes are already actively implemented in Lumen (Unreal Engine 5), RTXGI (NVIDIA Global Illumination), and in engines like Frostbite from EA.

So, combining direct method and ray tracing is not just a compromise, but a strategically correct architecture for the future of visualization. It allows you to flexibly adapt rendering to the current workload, scene, device type (desktop, console or mobile chip), while maintaining visual quality, control over computational costs and support for the latest physically based effects. This approach forms the basis of next-generation visualization standards.

Promising developments in ray tracing methods focus on achieving physical accuracy at high speed, reducing noise, and optimizing for real-time. Among the most relevant areas of development are the use of artificial intelligence, new sampling schemes, hybrid rendering, neural fields, and hardware innovations. One of the key achievements of recent years is the emergence of neural ray tracing, where tracing results are improved using deep neural networks. The most common example is neural denoisers such as NVIDIA OptiX or Intel Open Image Denoise, which allow achieving photorealistic images even with a very small number of samples. Neural methods are also used for importance sampling, approximation of lighting functions, and even for full scene reconstruction based on neural radiation fields, such as NeRF.

The most promising direction for real-time is ReSTIR (Reservoir-based Spatiotemporal Importance Resampling), which allows tracing only one or two rays per pixel, reusing information from previous frames and neighboring pixels. Its extension ReSTIR GI adds the ability to simulate indirect lighting, and ReSTIR PT – full path tracing with one or more reflections. By combining stochastic selection with adaptive accumulation of results, ReSTIR allows you

to reproduce global illumination in dynamic scenes with complex placement of light sources.

Adaptive Path Tracing is also actively developing, where the number of rays in each pixel depends on the local complexity of the scene – contrast, the presence of caustics or shadows. This allows you to effectively distribute computing resources, concentrating them in critical areas. At the same time, wavefront path tracing 2.0 allows you to optimize classic path tracing on the GPU, breaking the tracing stages into separate phases with effective processing in the form of vectorized waves (wavefronts), which allows you to reduce the branching of streams on the GPU and speed up tracing.

In the real-time sphere, the role of ray tracing with lighting caching (Radiance Caching) is growing, when the results of previous tracings are stored in buffers and reused in subsequent frames. This approach is widely used in the Unreal Engine 5 engine through the Lumen system, which provides soft indirect lighting and global effects with a minimum number of new traced rays. The direction of hybrid rendering is also promising, when geometry is processed by classical rasterization, and complex optical effects – mirrors, transparency, refraction – are traced separately, as needed. Some systems already implement automatic switching between methods depending on the complexity of the scene (adaptive hybrid rendering).

Neural Radiance Fields (NeRF) are attracting much attention, as they allow us to represent a scene not through geometric meshes, but as a continuous function parameterized by a neural network. Tracing in such a scene does not require the classical BVH structure and is performed through model inference for each ray. This approach is gradually being integrated with ray tracing for virtual reality visualization, 3D scanning, and photogrammetry tasks.

On the hardware side, next-generation RT cores are being developed (e.g., RT Core 3.0 in NVIDIA Ada Lovelace, Ray Accelerators in AMD RDNA3), which not only provide faster ray-tracing, but also support new mechanisms like Shader Execution Reordering (SER), which dynamically reorders computations to reduce branching in ray tracing. This improves efficiency when rendering scenes with diverse BRDFs, complex geometry, and shadows.

Another direction is adaptive sampling with learning, when the system, based on the previous frame or several frames, learns to determine which areas of the scene need more ray tracing. For this, reinforcement learning

methods, Bayesian models, or lightweight networks are used, which optimize the distribution of calculations in space and time. At the same time, denoising is also evolving – modern models based on transformers are better at restoring details in complex scenes, taking into account the global context, and are gradually replacing classic convolutional networks.

Thus, the future of ray tracing is a combination of classical physically correct models with statistical, stochastic, hybrid and neural approaches. They are integrated with hardware acceleration, adaptive mechanisms and new representations of the scene – from geometry to neural fields. In the coming years, ray tracing is expected to go beyond cinema and scientific visualization and become the standard not only for AAA games, but also for ordinary interactive environments, simulations, architectural design and educational applications.

Ray tracing techniques are the main tools for creating photorealistic images, as they allow you to model complex physical processes such as refraction, reflection and shadows. The choice between classical and physically correct ray tracing depends on the goals: classical ray tracing is suitable for simple scenes where you only need to define reflections and basic shadows, while physically correct ray tracing (PBR) is used to achieve more realistic results, especially when modeling materials and lighting. An important aspect is the use of global illumination, which includes modeling indirect light reflections and allows you to create more natural images where light reflects off objects in the scene and interacts with each other. To reduce the computational load when tracing rays, it is recommended to use methods such as spatial partitioning (for example, BVH – Bounding Volume Hierarchy), which allow you to speed up the search for ray intersections with objects, as well as apply adaptive tracing, which skips less important parts of the scene. Since ray tracing is computationally intensive, the use of parallel computing such as GPUs or technologies such as CUDA and OpenCL can significantly speed up the rendering process. In addition, for more realistic results, it is important to take into account the absorption and dispersion of light when working with transparent materials such as water or glass. Also important aspects are the management of noise in the final image, especially for methods with a low number of reflected rays, for which denoising or Monte Carlo methods can be used. To achieve physically correct results, it is necessary to configure materials with an accurate definition of the refractive

index and light scattering. In addition, additional effects such as depth of field, refraction of light through lenses and stereoscopic effect can significantly improve the realism of the final image. When implementing ray-tracing-based animations, it is important to ensure consistency of lighting and materials across frames, using techniques such as Motion blur and Temporal Anti-Aliasing (TAA) to achieve smooth animation and reduce artifacts. A consistent approach to testing and correcting the scene allows you to effectively optimize the rendering, gradually moving to more complex methods, which allows you to achieve the desired results without significant performance losses. Ray tracing is a powerful tool for achieving high-quality visualizations, but its effective use requires deep knowledge and optimization, especially when working with complex and realistic scenes.

To further optimize ray tracing methods, additional approaches can be used, such as breaking the scene into smaller parts or using multiple ray tracing to model more complex interactions between objects and light. This allows effects such as reflections from complex surfaces, micro-details (such as pores on the surface of materials), and diffuse scattering in materials that may have previously been missed due to computational limitations.

Memory and compute optimization is also an important aspect to maintain high performance with complex scenes. Instead of storing the entire scene geometry in memory, deferred compute techniques can be used, in which only the parts of the scene needed for rendering are stored in memory. This can reduce the load on the hardware, especially when working with large amounts of data.

Also, to ensure realistic and interactive scenes, dynamic lighting models can be implemented, allowing lighting to adapt in real time based on changes in the environment or user interaction. This allows lighting to be adapted to each specific scene, providing higher accuracy in modeling reflections, shadows, and other effects.

When implementing ray tracing techniques for real-time, it is important to consider the trade-offs between realism and performance. Using techniques such as pre-computed light maps or pseudo-realistic models (e.g., simplifying material physics) allows you to achieve a balance between high performance and realistic appearance without having to perform complex real-time calculations.

To achieve the best results, it is also important to constantly monitor new hardware capabilities, such as RT cores in modern GPUs (e.g. NVIDIA RTX), which significantly speed up the ray tracing process through hardware acceleration. In such cases, it is important to effectively integrate hardware capabilities to achieve maximum results in a shorter time.

In general, ray tracing methods require a careful approach to optimization and adaptation for specific tasks. The use of modern technologies, as well as constant improvement of algorithms and hardware, allows you to significantly improve the quality of visualizations, ensure high performance even in real time, and create photorealistic scenes in a variety of applications – from games to professional graphics and data visualization.

Ray tracing techniques can be very computationally intensive, so various simplification techniques are often used to achieve high performance. One of them is ray marching, which allows ray intersections with scene objects to be checked in stages, reducing the number of calculations and allowing the use of less detailed geometry, which is especially effective for scenes with a large number of small objects or textures. Spatial partitioning techniques such as Bounding Volume Hierarchy (BVH) or Octree are also used to reduce the number of intersection checks, which organize the scene into a data structure, allowing for faster discovery of potential objects for intersection and speeding up the ray tracing process. In addition, precomputed lightmaps and shadow maps allow you to store previously calculated lighting and shadow data, reducing the load on real-time calculations. To simplify physical material models, models such as Phong reflection can be used instead of physically correct reflection models, which can reduce computation time and maintain sufficient realism for certain scenes. Reducing the number of ray reflections is also an effective way to simplify. This can be achieved by setting a maximum number of reflections or fading the ray after a certain number of reflections, which reduces the number of rays traced and speeds up the rendering process. To model indirect light reflections, global illumination approximations such as radiosity or ambient occlusion can be used, which significantly reduces the computational cost compared to accurately reproducing each reflected ray. Using Selective Ray Tracing, where only certain rays are traced, such as from the camera to objects or important rays, can significantly reduce the load on the system. Instead of using ray tracing alone, it can be combined with other rendering techniques, such as rasterization, to achieve a balance between

speed and quality, allowing for faster rendering without significant loss of quality. In addition, denoising techniques can reduce the number of samples required to achieve high image quality, which is important in methods that use stochastic approaches, such as Monte Carlo rendering. Simplifying the shaders used to calculate the interaction of light with materials also allows for faster rendering, using less complex models that still provide sufficiently accurate results for many scenes. Each of these techniques can be applied depending on the image quality and performance requirements, and the choice of a specific strategy depends on the specifics of the project and hardware capabilities.

Ray tracing techniques are key to creating photorealistic images, allowing for accurate modeling of complex physical phenomena, while at the same time, with optimizations, providing high performance in modern rendering technologies.

DOI: 10.51587/9798-9917-51926-2025-023-9-30

Oleksandr ROMANIUK,
Doctor of Technical Sciences, Professor,
Vinnytsia National Technical University
ORCID ID: 0000-0002-2245-3364

Yuliya KOVALOVA,
Director of «Alpaca School»
ORCID ID: 0009-0009-6691-1353

WEB DESIGN DEVELOPMENT CONCEPTS

Website design[1] is an important step in creating an effective and attractive web resource that combines aesthetics, functionality and user-friendliness. It covers both visual and structural aspects that form the overall impression of the site. At the initial stage, design begins with studying the target audience, analyzing competitors and determining the main goal of the site. Based on the information collected, a visual style concept is created, which includes a color palette, typography, icons and other graphic elements that reflect the brand or idea of the project.

1 1Tidwell, J., Brewer, C., & Valencia, A. (2020). Designing interfaces: Patterns for effective interaction design (3rd ed., pp. 1–600). O'Reilly Media; Osborn, T. (2021). Hello web design: Design fundamentals and shortcuts for non-designers (140 p.). No Starch Press.