Computing and Artificial Intelligence 2025, 3(2), 2514.
https://doi.org/10.59400/cai2514

Article

An efficient ray tracing algorithm and its implementation based on adaptive octree decomposition

Chunlong Dong[1,*], Shengjun Xue[2], Limin Zhao[1]

[1] Silicon Lake College, Kunshan 215300, Jiangsu, China
[2] School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing 210044, Jiangsu, China
*** Corresponding author:** Chunlong Dong, dongchlsdu@163.com

CITATION

Dong C, Xue S, Zhao L. An efficient ray tracing algorithm and its implementation based on adaptive octree decomposition. Computing and Artificial Intelligence. 2025; 3(2): 2514. https://doi.org/10.59400/cai2514

ARTICLE INFO

Received: 6 January 2025
Accepted: 1 April 2025
Available online: 8 April 2025

Abstract: This paper proposes a ray tracing algorithm based on adaptive octree decomposition to solve the problem of low efficiency of calculating the intersection point of light rays and complex surfaces in the optical simulation of vehicle lights. The algorithm significantly improves the efficiency of solving the intersection problem by discretizing the complex surface into a series of polygonal facets and using a bilinear interpolation algorithm to optimize the light refraction calculation. Experiments show that the algorithm can greatly reduce the computation time on vehicle light models of different complexity, especially when dealing with complex surfaces, the algorithm improves performance by nearly 50%. The algorithm has been successfully applied to simulating optical performance in the design of vehicle lights and has achieved good application results, providing an efficient solution for the optical simulation in the design of vehicle lights.

Keywords: surface discretization; adaptive octree; ray tracing algorithm; optical simulation; performance improvement

1. Introduction

In order to meet the requirements for testing and evaluating optical properties in the process of designing vehicle lights, companies need an efficient optical simulation system that can quickly perform optical simulation and analysis after the product design is completed. With the system, companies can obtain the relevant optical performance data and optimize the product design. The geometrical model of vehicle lights is usually extremely complex and contains a large number of complex surfaces, which are often represented by spline surfaces in CAD systems, especially NURBS (Non-Uniform Rational B-spline) surfaces [1]. The current program directly employs Boolean operations between rays and spline surfaces to compute intersection points. However, this approach suffers from three critical limitations:

1) High computational complexity: Iterative ray-spline intersection calculations involve solving nonlinear equations with high parameter-space dimensionality, leading to scalability issues for large-scale models [2].

2) Redundant Computations: Determining the first intersecting surface requires checking all candidate surfaces and sorting distances, resulting in excessive memory usage.

3) Numerical Instability: In regions of high curvature or discontinuous geometry, iterative methods may fail to converge, compromising simulation accuracy.

Our proposed algorithm overcomes these challenges with two key inventive strategies.

1) Tolerance-controlled surface discretization: Complex spline surfaces are decomposed into planar facets within a preset geometric tolerance. This replaces computationally intensive spline-ray intersections with direct plane-ray intersections, eliminating iterative parameter-space searches.

2) Adaptive octree decomposition: A dynamic spatial partitioning strategy selectively refines regions of high surface density. By processing only sub-regions along the ray path, the algorithm minimizes redundant calculations and memory overhead.

Experimental results demonstrate that the algorithm reduces computation time by up to 50% for complex models (**Table 1**), effectively resolving the scalability challenges of traditional Boolean operations. By balancing geometric fidelity with computational efficiency, this approach provides a robust framework for high-performance optical simulations in automotive lighting design.

Table 1. Computation time comparison.

model complexity	Time taken before applying new algorithm (s)	Time taken after applying the new algorithm (s)	Performance improvement (%)
simpler	32	26	18.75%
moderate	605	451	25.45%
intricate	6652	3549	46.65%

2. Surface discretization

2.1. Planar approximation methods

By studying the key algorithms and techniques in the optical simulation process, and combining the principles of computer graphics on the ray tracing algorithm [3,4], it is known that in order to improve the computational efficiency of the ray tracing algorithm, the planar approximation method is used in this study. This method discretizes a complex surface into a series of combinations of planar facets that approximate the original spline surface within a preset tolerance [5,6].

Specifically, let S be a spline surface, Pi be a discretized planar facet, and ξ be a tolerance, then the discretization algorithm can be expressed as follows:

$$S \approx \sum_1^n Pi,$$

where n is the number of planar facets and for any i, Pi is satisfied:

$$\text{dist}\,(S, P_i) \leq \xi\,.$$

The intersection of a straight line with a plane is much faster than that of a spline surface, so the computation time can be significantly reduced by converting the complex intersection calculation of a straight line with a spline surface to that of a straight line with a plane. **Figure 1** shows the graphical effect of the surface before and after discretization.

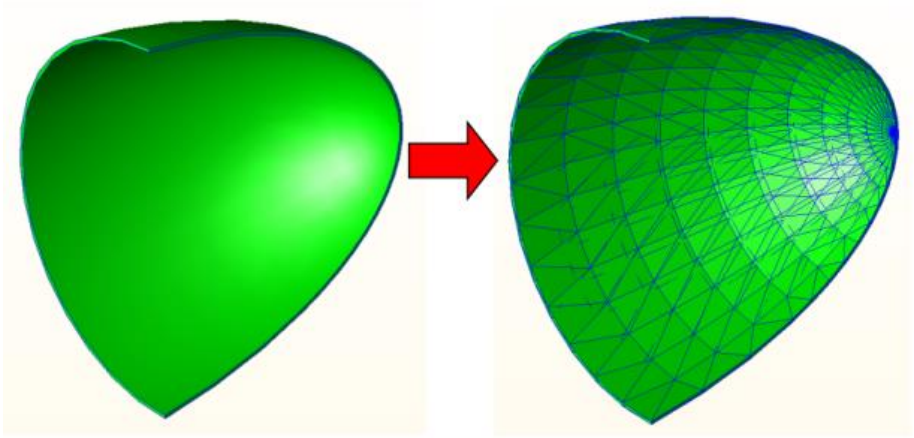

Figure 1. Surface discretization.

Figure 1 illustrates the comparison of surface geometry before and after discretization. Prior to discretization, the surface is represented as a continuous NURBS (Non-Uniform Rational B-Spline) surface. After discretization, the surface is approximated by a composite structure composed of multiple planar facets. In the pre-discretization state (left image), determining the intersection points of light rays with the continuous NURBS surface requires iterative solutions to complex equations involving parameters such as curvature and normal vectors, resulting in high computational complexity. In the post-discretization state (right image), the surface is decomposed into multiple planar facets, simplifying the ray-surface intersection calculation to direct solutions of plane equations, thereby significantly reducing computational time.

This discretization approach markedly reduces the complexity of ray-surface intersection calculations, enhancing computational efficiency. However, it introduces an inherent challenge: approximation errors. To mitigate excessive errors, the aforementioned tolerance parameter is employed to constrain the discretization process. The tolerance ensures that the deviation between the discretized planar facets and the original surface remains within an acceptable range, balancing computational efficiency with geometric accuracy.

The choice of tolerance ξ is crucial for the quality of surface discretization. Smaller tolerances can improve the accuracy of surface approximation, but increase the number of planar facets, thus increasing the computation amount; while larger tolerances may reduce the computation amount, but reduce the accuracy of surface approximation, which affects the accuracy of the final optical simulation results. In order to balance the computational efficiency and accuracy, it is necessary to choose the appropriate tolerance according to the specific application scenario.

In practical applications, the selection of tolerance needs to take into account a variety of factors. First, the tolerance should be determined according to the accuracy requirements of optical simulation. For example, in the design of vehicle lights, if higher accuracy of optical performance is required, a smaller tolerance should be chosen to ensure the accuracy of the simulation results. Second, the selection of tolerance should also consider the complexity of the model and the limitations of computational resources. For complex vehicle light models, a smaller tolerance may

lead to an excessive number of planar facets, thereby increasing the computational load and memory usage. Therefore, a balance needs to be struck between accuracy and computational resources when selecting the tolerance.

In our experiments, based on the geometric characteristics of automotive lamp models and the optical simulation requirements (surface error < 0.1 mm), we set an adaptive tolerance range of 0.05 mm to 0.2 mm. Specifically, stricter tolerances (0.05 mm) were applied to high-curvature regions like lens reflective surfaces for precise approximation. Conversely, relaxed tolerances (0.2 mm) were used for low-curvature areas such as lamp housings to optimize facet count. These parameters were determined through preliminary experiments on representative lamp models, balancing precision (average geometric deviation < 0.08 mm) and computational efficiency (about 30% fewer planar facets).

Moreover, the selection of tolerance can also refer to the geometric characteristics of the model. For regions with a larger curvature, a smaller tolerance is needed to ensure that the planar facets can better approximate the surface; while for regions with a smaller curvature, a larger tolerance can be chosen to reduce the number of planar facets. By adaptively adjusting the tolerance, the computational efficiency can be further optimized while ensuring accuracy [7].

2.2. Ray tracing optimization strategy

After surface discretization, the surface is modeled by a plane, which creates some error in the geometric representation and thus has some effect on the accuracy of the computed results After discretization, the light refraction effect in each small plane is considered to be the same, which has some error with the real surface refraction effect when it is not discretized. In order to reduce this error, an interpolation algorithm is used in this study to optimize the light refraction calculation.

The interaction of light rays with surfaces is a complex process, especially when it comes to the accurate calculation of normal vectors. In order to improve the accuracy of the refracted light ray calculation, this study uses a bilinear interpolation algorithm [8,9] to approximate the normal vector of the light ray incidence point on each planar facet.

Let the four vertices of the plane facet be $P1(x1, y1, z1)$, $P2(x2, y2, z2)$, $P3(x3, y3, z3)$ and $P4(x4, y4, z4)$. The corresponding normal vectors are $V1$, $V2$, $V3$ and $V4$ respectively. For any point $P(x, y, z)$ on the plane facet, its normal vector $V(x, y, z)$ can be calculated by bilinear interpolation:

$$V(x, y, z) = (1 - u)(1 - v)V1 + u(1 - v)V2 + (1 - u)vV3 + uvV4,$$

where u and v are the relative coordinates of the point P on the planar facet, which can be calculated from the coordinates of the point P and the coordinates of the vertices of the planar facet.

Figure 2 illustrates the refraction effect of light rays on discretized planar facets without employing an interpolation algorithm. As shown, within a single triangular facet, all rays exhibit identical refraction angles. Since the planar approximation serves as a simplified representation of the original curved surface, this approximation introduces deviations in the refraction direction of the rays compared to the actual

refraction behavior on the non-discretized curved surface. The discrepancy arises because the planar discretization homogenizes the normal vector across the entire facet, neglecting local curvature variations inherent to the original geometry. This inherent limitation underscores the necessity of integrating interpolation techniques to refine the accuracy of refraction calculations in ray tracing applications.

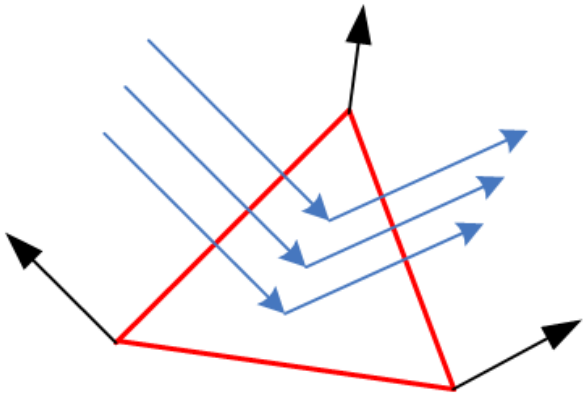

Figure 2. Refraction effect on discrete planes without interpolation.

Figure 3 shows the refraction effect of light rays on discrete planes after using the bilinear interpolation algorithm. With the interpolation algorithm, we are able to calculate the refractive direction of the light more accurately in each small plane, thus reducing the error due to plane discretization.

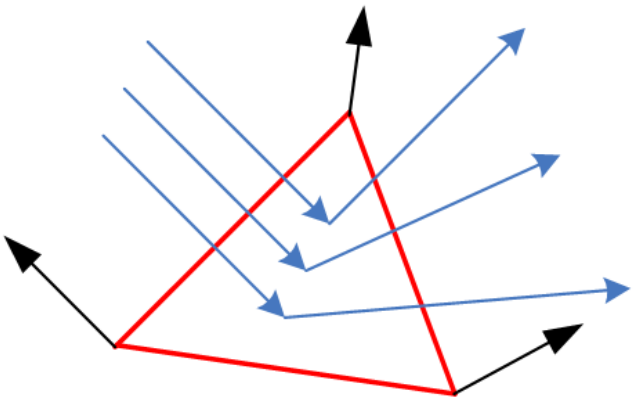

Figure 3. Refraction effect with bilinear interpolation algorithm.

In this paper, we quantitatively compared the performance of several interpolation algorithms in calculating light refraction. To intuitively show the advantages and disadvantages of different algorithms, we analyzed them using two key indicators: average refraction error and computation time. The specific results are shown in **Table 2**.

Table 2. Refraction error and computational time of interpolation algorithms.

Algorithm	Average Error	Computation Time (s)
Cubic Spline Interpolation	0.005	12.4
Nearest Neighbor Interpolation	0.015	5.2
Bilinear Interpolation	0.008	6.8

The tests were conducted on a platform equipped with an Intel Core i5-13500H processor, 16 GB of RAM, and an NVIDIA GeForce MX250 GPU. The test model, of medium complexity, comprised 20,000 planar facets. Ray tracing paths were computed over 100 iterations and averaged. Surface errors were quantified as the Euclidean distance between actual and interpolated surface normal.

Through experiments, we found that the bilinear interpolation algorithm achieved a good balance between computational efficiency and accuracy. Nearest neighbor interpolation [10] is simple to compute but has low accuracy, leading to significant deviations in the direction of refracted light in scenarios where high accuracy is required. Although cubic spline interpolation [11] has higher accuracy, its computational complexity is significantly increased, which greatly reduces the computation speed. In contrast, the bilinear interpolation algorithm, by utilizing the normal vector information of the four vertices of the planar facet, effectively controlled the computational load while ensuring a certain level of accuracy, making it more suitable for the application scenario of this study.

After the surface discretization, a bilinear interpolation algorithm is used to optimize the light refraction calculation by approximating the normal vector of the light incidence point on each planar facet in order to improve the accuracy of the refracted light calculation. The error due to plane discretization is reduced by the interpolation algorithm.

3. Adaptive octree decomposition algorithm

An octree structure is a hierarchical data model for partitioning three-dimensional space. Jackins and Tanimoto first proposed the use of the octree data structure to organize three-dimensional objects [12,13]. By recursively dividing the three-dimensional space into eight subspaces, each subspace can be further subdivided into smaller regions, thereby forming a hierarchical tree structure. This structure offers significant advantages in processing three-dimensional data, particularly in efficiently organizing and managing spatial data when dealing with complex geometric models and large-scale scenes. An octree consists of nodes, where each node represents a spatial region. The root node represents the whole space, and each child node corresponds to one-eighth of its parent's spatial region. Each node may contain one or multiple objects (e.g., geometric primitives, point clouds) and can be further subdivided into child nodes as needed. This hierarchical structure enables octrees to rapidly locate targets during spatial queries and object retrievals, avoiding the need to traverse the entire space. The spatial partitioning of an octree is axis-aligned bounding boxes, meaning the space is recursively split along the x, y, and z axes into eight subspaces. This partitioning method is simple, intuitive, and easy to implement. In practical applications, the granularity of partitioning can be adjusted based on specific requirements. For instance, in high-density data regions, spatial resolution can be improved by increasing the subdivision depth of the octree [14–17].

In ray tracing, the primary role of the octree is to accelerate the calculation of intersection points between rays and geometric objects in the scene. By dividing the space into multiple sub-regions, the octree efficiently narrows down candidate geometric objects that may intersect with the ray, eliminating the need to check every

individual object in the entire scene. This approach significantly reduces computational load and enhances rendering efficiency, especially in complex scenes [18,19]. The hierarchical structure of the octree enables highly efficient spatial queries. During ray tracing, when a ray intersects a sub-region of the octree, the algorithm skips sub-regions that do not intersect with the ray and only calculates intersections with geometric objects within the current sub-region. This spatial partitioning-based query method minimizes unnecessary computations, thereby enhancing the overall computational efficiency [20–22].

Despite the significant improvements in spatial query and intersection calculation efficiency provided by octrees in scenarios like ray tracing, practical applications still face challenges [23–25]. For example, when handling complex curved surface models, surface discretization subdivides a single curved surface into a large number of polygonal facets, drastically increasing the total number of facets in the model. This leads to higher computational loads when sorting ray-surface intersections. To address these issues, based on an in-depth study of the aforementioned literature, an adaptive octree decomposition algorithm is proposed.

3.1. Core idea of the algorithm

The fundamental principle of the adaptive octree decomposition algorithm is to decompose the spatial bounding box of the solid model into eight non-intersecting sub-bounding boxes using three planes perpendicular to each other, and if the number of surfaces intersecting with the sub-bounding box exceeds a predefined value, the sub-bounding box is subdivided into eight smaller bounding boxes. This recursive process eventually decomposes the solid box into an octree with different depths, with each subbox acting as a leaf node of the tree, and the number of surfaces intersecting with the sub-nodes not exceeding a pre-specified number of nodes.

The adaptive octree decomposition algorithm dynamically adjusts the spatial subdivision granularity based on the complexity of surfaces within bounding boxes. The key steps are as follows:

1) Initial subdivision

Action: Decompose the root bounding box B of the model into eight non-overlapping sub-bounding boxes B1, B2, ..., B8 using three orthogonal planes.

Purpose: Ensure uniform spatial partitioning and symmetry for efficient subsequent processing.

2) Surface Intersection check

Condition: For each sub-bounding box Bi:

If count (Bi) > T (where T is the predefined threshold):

Action: Mark Bi as requiring further subdivision.

else:

Action: Terminate decomposition and retain Bi as a leaf node.

3) Recursive subdivision

Process: For each Bi marked in Step 2:

Repeat Steps 1 and 2 on Bi until count (Bi) <= T or a maximum subdivision depth is reached.

The algorithm dynamically adjusts subdivision granularity, performing finer

subdivisions in complex regions (high surface density) to capture geometric details, while retaining coarser subdivisions in simple regions to minimize computational overhead.

In ray tracing using an adaptive octree [26,27], in order to find the first intersection of the ray with each surface in the object space, it is only necessary to sequentially detect those leaf nodes located in the ray's transmission path along the ray's extension direction. If there is an intersection of the ray with a surface in one of the leaf nodes on its path, the point closest to the ray is selected as the visible point; otherwise, the calculation continues with the next leaf node in the ray extension direction. On the premise that the ray is known to pass through a certain node, the next leaf node that the ray will pass through can be calculated directly. In this way, on the one hand, the leaf nodes that are not on the ray path do not have to be considered, and on the other hand, it is not necessary to sort the leaf nodes on the ray path, which narrows the search scope and reduces the amount of computation [28,29], and the schematic diagram of the decomposition of the adaptive octree is shown in **Figure 4**.

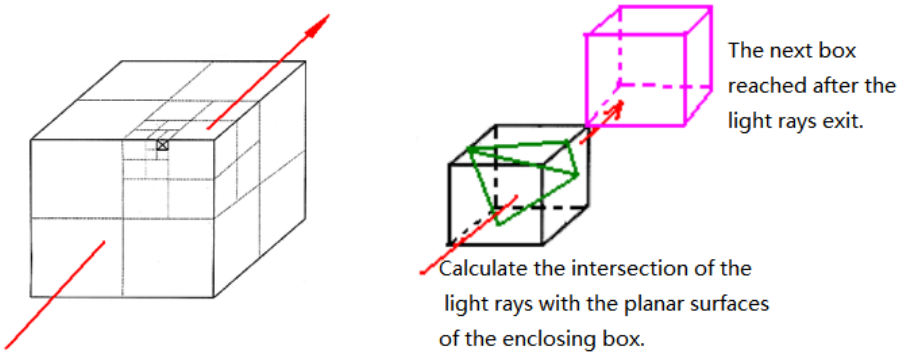

Figure 4. Schematic diagram of adaptive octree decomposition.

(Left) This subfigure demonstrates the three-dimensional spatial partitioning mechanism based on an octree structure. The root node (representing the bounding box of the entire scene) generates sub-cubes through recursive subdivision. The color intensity of the nodes reflects regional complexity: darker regions indicate high-density surface distributions, whereas lighter regions correspond to geometrically sparse structures. The adaptive subdivision strategy dynamically adjusts the granularity of partitioning based on the number of original geometric facets contained within each sub-node. Complex regions (e.g., multi-cavity structures in vehicle lamps) are subdivided into smaller cubes to precisely capture detailed features, while simpler regions maintain coarser subdivisions to optimize computational resources.

(Right) This subfigure illustrates a typical ray tracing path. The ray originates from the viewpoint and sequentially performs intersection detection with the bounding boxes of nodes at various octree levels. When a ray intersects a node's bounding box, the algorithm dynamically determines whether to trace its child nodes based on the geometric information stored in that node. This process continues until the ray intersects the actual geometric facets within a leaf node. The final intersection point is then utilized for subsequent shading calculations, completing the ray tracing procedure.

Through analysis, we can see that the advantages of using the adaptive octree

decomposition method are as follows:

1) Only the intersections of the light rays with the sub-surrounding boxes containing the facets on their transmission paths are computed.

2) Sorting the bounding box according to the direction of light transmission without sorting the polygonal facets.

3) Applicable to non-uniform models with complex geometry.

To more clearly describe the adaptive octree decomposition algorithm, we provide the following pseudocode:

```
function Adaptive Octree Decomposition (B, T):
if count(B) <= T:
return B
else:
B1, B2, ..., B8 = Split(B)
for each Bi in [B1, B2, ..., B8]:
Adaptive Octree Decomposition (Bi, T)
```

Here, count(B) represents the number of surfaces intersecting with the bounding box B, and Split(B) denotes the operation of decomposing the bounding box B into eight sub-bounding boxes.T is a threshold that determines when to stop further subdividing the bounding boxes.

Regarding the algorithm's complexity, let the edge length of the model space bounding box be L, the edge length of the initially divided sub-bounding box be l, the total number of surfaces be *N*, and the threshold be *T*. In the worst case, the algorithm's time complexity is O(NlogN), and the space complexity is O(N). The specific derivation is as follows:

Time complexity: Each decomposition operation divides the bounding box into eight sub-bounding boxes, and at most log8(N) decomposition operations are needed. For each sub-bounding box, the complexity of the intersection calculation operation is O(N), so the total time complexity is O(NlogN).

Space complexity: In the worst case, each sub-bounding box may become a leaf node, so the space complexity is O(N).

Compared with the traditional ray tracing algorithm, this algorithm effectively reduces the computational amount of ray and surface intersection and reduces the time complexity by dividing the adaptive octree. For example, when dealing with a complex headlight model containing surfaces, the computation time of the traditional algorithm grows exponentially with the number of surfaces, while the time complexity of this algorithm grows slowly, which is a more significant advantage in large-scale data.

3.2. Execution flow of the algorithm

In order to clearly demonstrate the specific execution steps of the adaptive octree decomposition algorithm, a flowchart of the algorithm is provided in **Figure 5**. This flowchart provides a visual understanding of the recursive decomposition mechanism of the algorithm and how it can be applied in ray tracing. The following is an overview of the execution flow of the ray tracing algorithm based on the adaptive octree decomposition, which focuses on the tracing process of a single ray.

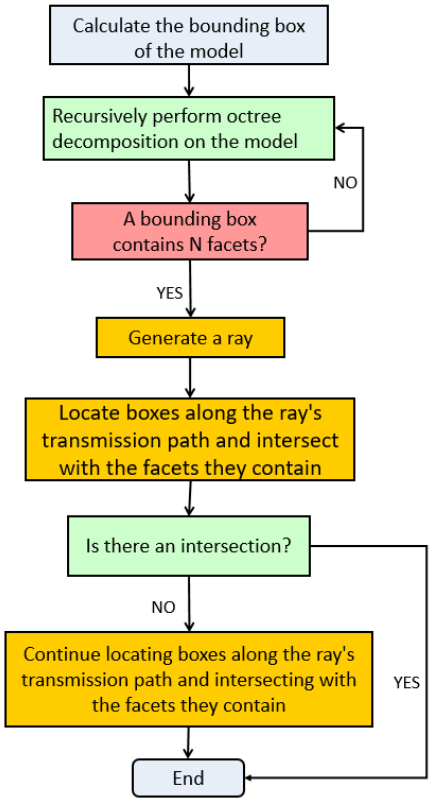

Figure 5. Flowchart of ray tracing algorithm based on adaptive octree decomposition.

1) Compute the bounding box of the model:

The algorithm first computes the spatial bounding box of the entire model, which serves as the basis for the subsequent octree decomposition.

2) Recursively decompose the solid model into an octree:

Next, the algorithm recursively decomposes the model to generate an adaptive octree. This step is the core of the algorithm, which divides the model space into multiple subspaces through recursive decomposition, with each subspace containing no more than the preset threshold number of facets.

3) Check if the bounding box contains facets:

During the recursive decomposition process, the algorithm checks whether each bounding box contains facets. If it does, the decomposition continues; if not, the decomposition stops.

4) Generate a ray:

At the start of ray tracing, the algorithm generates a ray, which will be used for subsequent intersection calculations.

5) Find the boxes on the ray's path and intersect with their contained facets:

The algorithm searches along the ray's path for bounding boxes that intersect with the ray and calculates the intersections between the ray and the facets contained within these bounding boxes.

6) Check for actual intersections:

For each potential intersection point, the algorithm verifies whether a true intersection exists. If an intersection is found, the algorithm records this point; if not, it continues to search for the next potential intersection point.

7) Termination:

The algorithm terminates when it finds the first intersection point between the ray and the model or when the ray exits the model's range.

Through the above steps, the adaptive octree decomposition algorithm effectively optimizes the computational efficiency of the ray tracing process, especially when dealing with models containing a large number of complex surfaces. By reducing unnecessary calculations and precisely controlling the scope of computation, the algorithm significantly enhances the speed and accuracy of ray tracing.

4. Program design scheme

4.1. Program architecture design

The program module will be developed using C++ with an object-oriented design approach, and in order to optimize the design and algorithms for data structures and data storage, STL (Standard Template Library) and the BOOST and OPENGL graphic libraries will be used as auxiliary development packages [30–32]. If the discrete functions of the surfaces are to be performed in a program module rather than in CATIA, graphics programming will be performed with the help of ACIS and other graphics development packages. The software will be structured as a stand-alone program or as a Dynamic Link Library (DLL) to interact with CATIA and other CAD programs [33]. In terms of data structure, depending on the design of the program, it is necessary to extend some of the original classes or add new classes and data. The program architecture of this optical simulation algorithm is shown in **Figure 6** below:

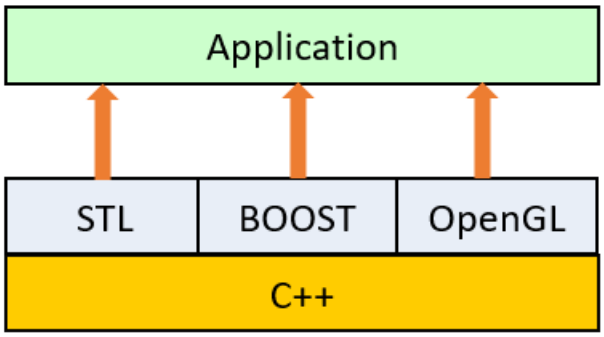

Figure 6. Basic architecture of the program.

4.2. File interface and file format

In this program design, a model conversion tool based on the secondary development of CATIA is used to convert the CAD model of the headlight into an input file for the ray tracing program. Use the interface to read the file into the optical calculation program module, optical performance simulation calculations, after completing the relevant calculations, the calculation results file saved. And use the visualization tool developed based on HOOPS rendering engine to visualize the calculation results, the specific interaction flow is shown in **Figure 7** below:

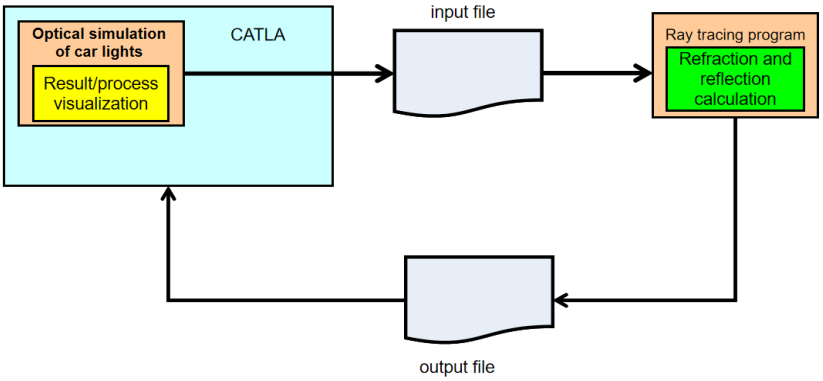

Figure 7. Schematic diagram of file interaction.

The input file is divided into two parts: the definition of the light source and the definition of the solid model, which is represented by a series of planar facets due to the discretization of the surface into planes. The output file is also divided into two parts, one of which is the statistical light data received by the receiving surface, which is represented as a two-dimensional array according to the defined grid size of the receiving surface, and the other is the light data during the optical simulation process, which is used to visualize the optical simulation process.

Input file Data_1: (definition of light sources)
Point3D x0, y0, z0 (launch point)
Vector3D nx, ny, nz (emission vectors)
iNumofLine (ray number)
Ratio_Refraction (refractive index of the medium)
iNumofRefraction (number of refractions)
.....
Input file Data_2: (definition of entity model)
Facet
iNumofFacet (number of facet)
Vector3D nx, ny, nz (normal vectors of faces)
outer loop:
    Point3D x1, y1, z1
    Point3D x2, y2, z2
    Point3D x3, y3, z3
End loop
Ratio_Refraction
.....
end facet
Output file Data_3: (receiving surface statistics, analog light data)
Results [n][n] (Receiving surface statistics)
Ray3D [n] (analog ray list)

4.3. Program flow

The program flow of this optical calculation program module is shown in **Figure 8** below:

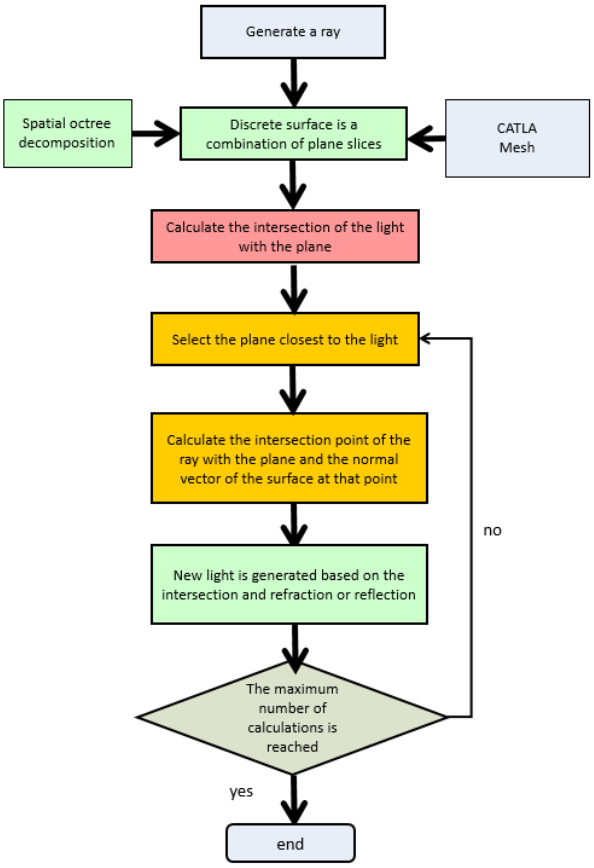

Figure 8. Program flowchart.

According to **Figure 8**, the program involves three modules during the whole running process, the surface discretization module, the optical calculation module and the visualization display module. The three modules are independent of each other in the program design process, and can communicate through files or direct memory communication. The surface discretization module and the result visualization module can be based on some commercial software for secondary development, or based on the ACIS/HOOPS graphics rendering engine [34] for independent development.

5. Experimental analysis

5.1. Experimental design

In order to verify the effectiveness of the ray tracing algorithm based on adaptive octree decomposition proposed in this paper in practical applications, the performance of the software before and after applying the algorithm is tested and compared. Three types of lamp models with different complexities were used in the experiments, representing geometrically simple single-cavity lamps, lamp bodies with medium complexity, and geometrically complex multi-cavity lamp bodies, respectively. The computation time of the software was recorded by using an ordinary personal computer with the same computer hardware configuration. A geometric model of a multi-cavity lamp body is shown in **Figure 9** below.

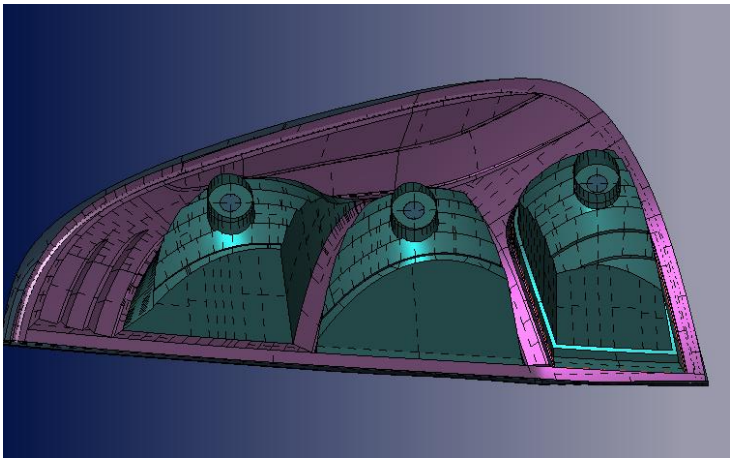

Figure 9. Geometric model of multi-cavity lamp body.

Figure 9 illustrates the geometric model of a multi-cavity lamp body, which represents a category of vehicle lights characterized by intricate geometric structures. The model features multiple cavities, with relatively complex design details and internal configurations. Such models impose stringent demands on the efficiency and accuracy of algorithms in ray tracing computations due to their geometric sophistication. By testing the software's performance on this model using standard personal computers, the evaluation highlights the capability of the proposed algorithm to handle geometrically complex structures. This approach provides a practical means to assess the advantages of the new algorithm in scenarios requiring high computational precision and resource optimization for complex geometric models.

5.2. Experimental results and analysis

Before applying the new algorithm, the software took 32 s to process a geometrically simple single-cavity lamp, 605 s to process a lamp with moderate complexity, and a whopping 6652 s to process a geometrically complex lamp containing complex surfaces. However, after integrating the ray tracing algorithm based on adaptive octree decomposition, the performance is significantly improved when processing the same models. The time taken to process a geometrically simple single-cavity lamp is reduced to 26 s, for lamps with medium complexity the time becomes 451 s, and for lamps with complex geometry containing complex surfaces the time is drastically reduced to 3549 s, which is a performance improvement of almost 50%.

This comparison clearly shows that the algorithm proposed in this study has a significant effect on the computational efficiency of ray tracing, especially when dealing with complex surface models, which greatly reduces the computation time and effectively solves the problem of inefficient computation of the intersection of rays and complex surfaces, and provides a more efficient solution for the optical simulation in the design of vehicle lights.

The aforementioned experiments also have certain limitations. For instance, the number and types of experimental models may be limited and cannot fully represent all scenarios in practical application contexts. Additionally, the experimental environment may differ from some actual production environments. Therefore, in

future research, we will further expand the scale of experiments, consider a wider variety of vehicle light models, and validate the universality and stability of the algorithm on different computing platforms.

6. Conclusion and future work

The ray tracing algorithm based on adaptive octree decomposition proposed in this study shows significant efficiency improvement in the field of vehicle light optical simulation. By discretizing complex surfaces into a combination of planar facets and optimizing the light refraction computation with a bilinear interpolation algorithm, the algorithm is able to significantly reduce the computation time when dealing with headlight models of different complexity, and in particular, the performance is improved by nearly 50% when dealing with complex surfaces. The experimental results show that the algorithm outperforms the existing techniques in terms of computational efficiency and accuracy, and provides an efficient solution for optical simulation in headlight design.

Future work will explore multi-threading or parallelization of the design to further improve computational speed [35–37], and consider system resource optimization independent of CAD software to improve program versatility and efficiency. In addition, considering the actual production situation and computer operation, the following aspects can be considered to improve the computational efficiency.

1) Multi-threaded or parallelized program design. Ray tracing is suitable for multithreaded or parallelized design, and multithreaded parallelized design methods are considered at the time of program design to increase the computational speed.

2) Independent of CAD software to optimize system resource consumption. That is, the above-mentioned surface discrete module, optical simulation module, and calculation results display module in the design and development consider the use of independent development, not based on the secondary development of some commercial CAD software, so that the program does not rely on other CAD software, on the one hand, can be managed in memory, optimizes the program's resource consumption, and improves the speed of calculation. On the other hand, it also facilitates the expansion to other CAD platforms, separating design and analysis.

3) Comprehensively optimize the program design architecture and memory management, including optimizing module interaction logic, adopting dynamic memory allocation optimization and cache strategy optimization, and other techniques to reduce memory fragmentation and data read/write latency; screen and integrate reliable and stable third-party libraries, and make use of their high-efficiency algorithmic implementations and optimized data operation interfaces to enhance the overall performance of the program and to ensure that the data operation and arithmetic methods achieve the optimal state [38].

Future work will also address potential challenges such as ensuring algorithm stability across diverse hardware platforms and optimizing memory usage for large-scale models.

Author contributions: Conceptualization, CD and SX; methodology, CD; software,

LZ; validation, CD, SX and LZ; formal analysis, CD; investigation, CD; resources, LZ; data curation, LZ; writing—original draft preparation, CD; writing—review and editing, SX; visualization, LZ; supervision, CD; project administration, CD; funding acquisition, SX. All authors have read and agreed to the published version of the manuscript.

Institutional review board statement: Not applicable.

Informed consent statement: Not applicable.

Conflict of interest: The authors declare no conflict of interest.

References

1. Zhao F. Key technologies and applications for cloud CAD software. Computer Integrated Manufacturing Systems. 2022; 28 (04): 959-978. doi: 10.13196/j.cims.2022.04.001
2. Wang Y, Lyu H, Chen X. An Efficient Method for Computing the intersection between Two B-Spline Curves. Journal of Computer-Aided Design & Computer raphics. 2024; 36(05): 687-700. doi: 10.3724/SP.J.1089.2024.19869
3. Xu X, Wu X, Chen Z, et al. A Survey of Ray Tracing Rendering of Large-Scale Scenes. Journal of Computer-Aided Design & Computer Graphics. 2024; 36(8): 1155-1170. doi: 10.3724/SP.J.1089.2024.2023-00614
4. Qin Z, Zhang W, Jiang X, et al. Real-time Interactive Computer-generated Integral Imaging Method Based on Ray Tracing. Acta Photonica Sinica. 2019; 48(9): 0911002. doi: 10.3788/gzxb20194809.0911002
5. Yan R, Huang L, Guo H, et al. Review of Real-Time Ray Tracing Technique Research. Journal of Frontiers of Computer Science and Technology. 2023; 17(02): 263-278. doi: 10.3778/j.issn.1673-9418.2207067
6. Jin Y, Song D, Yu C, et al. Curve Design Method on Mesh Surface Based on Distance Constraints. Journal of Software. 2020; 31(10): 3266-3279. doi: 10.13328/j.cnki.jos.005804
7. Zhao Z. Developable Approximations for Discrete Surfaces. University of Science and Technology of China; 2023. doi: 10.27517/d.cnki.gzkju.2023.000196
8. Xiao H. Bilinear Interpolation Parallel Algorithm Based on GPU Computing. Journal of C hinese ComputerSystems. 2010; 31(11): 2241-2245. doi: 10.20009/j.cnki.21-1106/tp.2010.11.023
9. Chen L, Gao C. Fast discrete bilinear interpolation algorithm. Computer Engineering and Design; 2007. doi: 10.3969/j.issn.1000-7024.2007.15.077
10. Tang W, Qi S, Yang X, et al. Traffic Flow Data Repair Method Based on Random Forest and Nearest Neighbor Interpolation. Science Technology and Engineering. 2024; 24(32): 14056-14065. doi: 10.12404/j.issn.1671-1815.2307559
11. Guo Y. Research on Path Planning and Autonomous Obstacle-avoidance Algorithm for Unmanned Vehicle. Beijing Jiaotong University; 2021. doi: 10.26944/d.cnki.gbfju.2021.001488
12. Jackins CL, Tanimoto SL. Oct-tree and their use in representing three-dimensional objects. Computer Graphics & Image Processing. 1980; 14(3): 249-270. doi: 10.1016/0146-664X(80)90055-6
13. Liu AA, Nie WZ, Su YT. 3D object retrieval based on multi-view latent variable model. IEEE Transactions on Circuits and Systems for Video Technology. 2019; 29(3): 868-880. doi: 10.1109/TCSVT.2018.2810191
14. Zhang M, Yan M, Ma Y, et al. 3D Voxel Model Retrieval Based on octree Structure. Chinese Journal of Computers. 2021; 44(2): 334-346. doi: 10.11897/SP.J.1016.2021.00334
15. Jia S, Zhang W, Yu X, et al. Parallel Collision Algorithms for Cutting Simulation of Octree-based Deformable Objects. Journal of Computer-Aided Design & Computer Graphics. 2017; 29(12): 2180-2188. doi: 10.3724/SP.J.1089.2017.16430
16. Wang Y, Tan S, Jing W, et al. NURBS surface reconstruction based on space partitioning of octree. Computer Engineering and Design. 2015; 36(06): 1565–1570. doi:10.16208/j.issn1000-7024.2015.06.031
17. Zhang P, Du C, Zhao W. Study on mesh generation algorithm of polyhedron in 3D scaled boundary finite element method based on image octree. Journal of Hohai University (Natural Sciences). 2020; 52(1): 46-54. doi: 10.3876/j.issn.1000-1980.2024.01.006
18. Zhou H, Wu C. Ray tracing algorithm based on neighborhood search of octree spatial encoding. Computer Engineering and Design; 2017. doi: 10.16208/j.issn1000-7024.2017.01.050

19. Zhang W, Xie S, Zhong J, et al. Acceleration algorithm in ray tracing by the octree neighbor finding. Journal of Graphics. 2015; 36(03): 339-344. doi: 10.3969/j.issn.2095-302X.2015.03.003
20. Cai X, Zeng L, Liu G. Survey of ray tracing in volume rendering. Computer Engineering and Design. 2009; 30(21): 4956-4959. doi: 10.16208/j.issn1000-7024.2009.21.027
21. Xiao W, Zhang L, Sun D, et al. Ray-tracing based occlusion detection algorithms. Acta Geodaetica et Cartographica Sinica. 2014; 43(8): 862-868. doi: 10.13485/j.cnki.11-2089.2014.0134
22. Hou L, Liu Y, Wang Y. Study of collision detection based on ray tracing principle. Journal of System Simulation. 2006; 18(Suppl.1): 84-87. doi: 10.16182/j.cnki.joss.2006.s1.027
23. Cai P. Rendering of point cloud data based on ray tracing and photon mapping. Beijing University of Technology; 2013. doi: CNKI:CDMD:1.1013.047307
24. Cai P, Yin B, Kong D. A ray tracing method based on the nearest points. Journal of Graphics. 2013; 34(3): 1-6. doi: 10.3969/j.issn.2095-302X.2013.03.001
25. Zheng Y, Xie Y, Ji Yu et al. 3D cloud simulation based on octree neighbor finding of ray tracing. Computer Engineering and Design. 2023; 41(5): 1489-1493. doi: 10.16208/j.issn1000-7024.2020.05.043
26. Yao C, Ma C. Adaptive octree 3D image reconstruction based on plane patch. Optics and Precision Engineering. 2022; 30(9): 1113-1122. doi: 10.37188/OPE.20223009.1113
27. Liu X, Weng X, Chen Hao, et al. An Improved Algorithm for Octree-Based Exact Collision Detection. Journal of Computer-Aided Design & Computer Graphics. 2005; 17(12): 2631–2635. doi: 10.3321/j.issn:1003-9775.2005.12.008
28. Wang X. Octree algorithm for point surface matching. Computer Engineering and Applications; 2009. doi: 10.3778/j.issn.1002-8331.2009.32.054
29. De Queiroz RL, Garcia DC, Chou PA, et al. Distance-basedprobability modelforoctreecoding. IEEE Signal Processing Letters. 2018; 25(6):739-742. doi: 10.1109/LSP.2018.2823701
30. Zhang D. Development of CAE model interaction and icon rendering engine based on OpenGL. Hunan University; 2023. doi: 10.27135/d.cnki.ghudu.2023.000451
31. Xu Z, Zhang Z, Ding Y. Application of Boost Library to analysis of geographical network. Engineering of Surveying and Mapping. 2010; 19 (03): 55-58. doi: 10.3969/j.issn.1006-7949.2010.03.016
32. Zheng Y, Zheng P. Topology reconstruction of STL data based on C++ standard template library. Chinese Journal of Engineering Design. 2013; 20(06): 522-528. doi: 10.3785/j.issn.1006-754X.2013.06.013
33. Zhang Z. Research on Die Forging Process CAD System and Key Technologies for Large Aircraft Component Based on CATIA. Huazhong University of Science & Technology; 2016. doi: 10.7666/d.D01077044
34. Meng X, Zhang L, Xiao T. Research on Assembly Simulation System Based on ACIS and HOOPS for Complex Products. Journal of System Simulation. 2014; 26(10): 2381-2385. doi: 10.16182/j.cnki.joss.2014.10.092
35. Shang M, Zheng Y, Chen J, et al. A multi-threaded parallel algorithm for quality improvement of tetrahedral meshes. Chinese Journal of Computational Mechanics. 2016; 33(4): 613-620. doi: 10.7511/jslx201604030
36. Wang F, Sun Z, Zheng X, et al. Discretizing approach for complex three-dimensional geometry based on meshless particle methods. Journal of Zhejiang University (Engineering Science). 2022; 56(08): 1597-1605. doi: 10.3785/j.issn.1008-973X.2022.08.014
37. Sun Y, Luo S, Hu X, et al. Modeling of bulge formed join intersection region based on differential equation surface intersection tracking method. Chinese journal of construction machinery. 2024; 22(02): 141-145. doi: 10.15999/j.cnki.311926.2024.02.022
38. Yu R. Research of Memory Management Mechanism Based on Dynamic Allocation Algorithm. Harbin Engineering University; 2017. doi: CNKI:CDMD:2.1018.081593