Enhancing Real-Time VR Scene Rendering with Optimized Task Path Scheduling Via Dynamic Programming Techniques

Jing Ma
College of nursing health, Yunnan Open University, Kun'ming650500, Yunnan, China
E-mail: majing140627@163.com

Keywords: dynamic programming algorithm, real time rendering, virtual reality, performance optimization

Received: June 11, 2025

This article proposes a dynamic programming rendering path optimization algorithm for virtual reality scenes, which innovatively introduces real-time feedback mechanism and sliding window resource scheduling, effectively improving rendering performance under multi scene concurrency conditions. Compared with traditional greedy, genetic, and local search methods, this method can achieve global optimal path selection and has been experimentally verified to achieve significant improvements in key indicators such as average frame rate and resource utilization. All experiments in this article were independently repeated 5 times, and the results were tested for variance and significance. This article proposes for the first time a dynamic programming path scheduling strategy that combines sliding window real-time feedback, solving the problem of generating the global optimal path in multi task high concurrency environments. Compared with traditional heuristic and intelligent optimization algorithms, the proposed method significantly improves system stability and rendering performance, and has stronger practicality and promotional value. All variables and parameters are clearly defined in the Methods section, and the experimental process and evaluation criteria follow internationally recognized standards to ensure the reproducibility and rigor of the conclusions. The system conducted experiments on three types of virtual environments: typical city blocks, natural terrain, and indoor exhibition halls on the Unity platform. The results showed that after introducing dynamic programming, the average frame rate increased from 76.9 FPS to 86.3 FPS, GPU utilization increased from 85.1% to 91.6%, task completion rate increased from 92.8% to 98.2%, and rendering failure rate decreased from 3.1% to 0.9%. The solution significantly improves the real-time performance, stability, and resource utilization efficiency of the system while ensuring image quality. Compared with traditional static scheduling methods, this method exhibits better response capability and scalability in multi task high concurrency environments, providing efficient technical support for complex virtual reality rendering tasks.

Povzetek: Za izboljšanje realnočasnega izrisovanja VR-prizorov je razvit algoritem dinamičnega programiranja (DP-RPS), ki združuje drsno okno in sprotno povratno zanko za optimizirano razporejanje poti nalog. Na platformi Unity doseže zelo dobre rezultate. Zagotovi stabilno, prilagodljivo in energetsko učinkovito izrisovanje VR-prizorov z globalno optimizacijo poti pri večopravilnosti in visoki sočasnosti.

1 Introduction

With the widespread application of virtual reality technology in games, movies, healthcare, architecture, and other fields, users' demands for immersive experience and response speed are constantly increasing, which drives the evolution of rendering systems themselves to achieve high efficiency and intelligence. In 3D scenes with massive details, real-time rendering systems require graphic rendering and final result presentation in a short amount of time, ensuring frame rate while controlling resource fluctuations and delays. However, traditional solutions based on fixed pipelines or static scheduling are no longer sufficient to meet the needs of multi task parallelism, heterogeneous inputs, and environmental changes. It is necessary to construct a new intelligent scheduling system.

Although some improvements have been made in image compression and hardware acceleration, the constraints of resource allocation, execution path, and task sequence often result in frame rate jitter and delay in the interactive environment. Therefore, this study adopted dynamic programming techniques to re optimize the scheduling strategy and improve the overall efficiency of the system. Xionghui et al. pointed out in their study of interactive virtual systems that the consistency of task paths and the accuracy of feedback links directly affect immersion and operational response quality [1], thus verifying the impact of path optimization on real-time performance. Lumpkin T L et al.'s research shows that familiarity, aesthetics, and

change type affect visual memory in VR, influencing system frame rate stability and load control[2], which reflects the important role of real-time task scheduling mechanism in system performance optimization.Sainz M's research shows that accelerating real-time global illumination rendering can effectively improve the system's response speed and image realism, thereby enhancing user immersion and interactive experience [3],Further verification has shown that the rational optimization of routing paths and task scheduling strategies is a critical requirement for this system to have real-time and stable feedback.

By constructing task dependency structures through state transition models, dynamic programming can generate optimal rendering paths that balance frame rate and stability. The virtual behavior modeling method proposed by Htun N S et al. [4] provides a supporting framework for task prediction and state control in this study. Compared to heuristic algorithms, dynamic programming has global optimization and real-time correction capabilities, making it suitable for high-density interactive scenarios.

Although there have been studies using dynamic programming and other optimization algorithms for virtual reality rendering scheduling, there are often issues with insufficient global optimality or lack of real-time feedback mechanisms. This article constructs a dynamic programming state transition model and introduces sliding window feedback adjustment to achieve global optimization and dynamic adaptation of path planning, significantly improving rendering frame rate and resource utilization.

This study starts from two dimensions: algorithm construction and system integration, designs task nodes and path mechanisms suitable for VR rendering processes, and implements closed-loop scheduling optimization in the platform.Owens J D et al.'s research shows that GPU computing technology significantly enhances large-scale parallel processing capabilities, providing strong support for high-performance computing and real-time rendering, thereby promoting the development of resource scheduling mechanisms for complex systems [5].

The goal of this study is to build an embeddable and multi scene adaptive real-time rendering optimization framework, with dynamic programming as the core, to solve the bottleneck problems in current virtual reality rendering and provide stable and efficient technical support for image generation systems.Based on the above analysis, this study mainly focuses on the following two core issues: (1) How to optimize the rendering path scheduling of virtual reality scenes based on dynamic programming, and achieve real-time and stability improvement under high concurrency and high complexity tasks? (2) Can the proposed scheduling algorithm achieve significant optimization in core metrics such as frame rate and GPU utilization compared to existing mainstream methods in different typical scenarios?

2 Related work

Virtual reality systems require extremely high real-time rendering performance, and rendering efficiency has become a key factor affecting the system's interactive experience. Early technological methods mainly relied on fixed pipeline mechanisms, which achieved frame by frame generation of graphic images through static task sequential execution. However, in scenarios with high task intensity and frequent perspective switching, the fixed path mode is difficult to fully utilize computing resources, often resulting in the coexistence of delay fluctuations and resource redundancy.

To improve rendering performance, scheduling has become one of the optimizations focuses. Heuristic strategies are widely used in early scheduling design, such as priority-based sorting mechanisms, local greedy methods, and task complexity estimation models. These methods respond quickly in scenarios with light computational burden, but due to their local optimization properties, they lack dynamic perception and regulation of the global state, making it difficult to run stably in complex environments such as high concurrency and asynchronous input.

To overcome the above bottlenecks, some strategies integrate intelligent optimization methods such as genetic algorithm, ant colony algorithm, simulated annealing, etc., using global search capabilities to avoid the problem of path getting stuck in local optima. Although this type of method improves path quality, it relies heavily on computing power, has complex parameter tuning, and an unstable training process, which limits its efficiency in systems with high real-time requirements.Li B et al.'s review pointed out that the application of deep learning technology in virtual reality and augmented reality significantly improves the perception ability and interaction efficiency of the system, providing theoretical support for the structured design and feedback mechanism of complex tasks [6].

The dynamic programming algorithm demonstrates advantages in multi-stage path decision-making through state recursion and sub problem optimal solution mechanisms. By constructing a state transition network and cost function model, the rendering process can be decomposed into ordered subtasks, and the optimal path can be calculated layer by layer while recording intermediate results.The image processing pipeline optimization method proposed by Ragan Kelley J et al. achieves flexible and efficient image processing performance improvement by decoupling the algorithm from scheduling, providing technical support for the construction of complex tasks in virtual environments [7].Chae L R et al. pointed out that the humanoid level of virtual characters significantly affects users' trust and acceptance, which in turn affects users' tolerance for system latency. This provides a psychological reference for behavior perception embedding in scheduling mechanisms [8]. The existing optimization strategies have their own characteristics in path generation logic, scheduling objectives, system response, and other aspects. To compare the adaptability and limitations of different methods more clearly, Table 1 summarizes and organizes the mainstream rendering scheduling optimization methods:

Table 1: Comparison of mainstream rendering scheduling optimization methods

Method Type	Core mechanism	advantage	limitation	Applicable scenarios
heuristic strategy	Rule based local path selection	Simple implementation and fast speed	Easy to fall into local optima and lack global control	Low complexity, fixed task flow
intelligent optimization algorithm	Genetic algorithm, ant colony, simulated annealing, etc	Strong global search capability	Complex parameter tuning and high computational cost	Medium complexity and offline scheduling scenarios
reinforcement learning model	Strategy learning based on reward feedback	Can adapt to some dynamic task changes	Long training time and weak generalization ability	A small amount of dynamic and highly trainable environment
Graph Neural Network	Representation and Propagation Learning Based on Graph Structure	Capable of modeling complex task dependencies	Strong data dependency, difficult to migrate	Clear diagram structure, stable task environment
Dynamic Programming	Optimal combination strategy of state transition and subproblem	Path controllability, global optimization, strong stability	High initial modeling requirements and structured representation of tasks	Complex scenarios with multiple dependencies and optimal path requirements

Overall, current mainstream rendering scheduling methods such as heuristic and intelligent optimization algorithms have certain performance advantages in some scenarios, but are susceptible to factors such as local optima, parameter sensitivity, and poor generalization, making it difficult to meet the system requirements of high concurrency, complex dependencies, and real-time performance. The dynamic programming strategy introduced in this article has the advantages of global path search and structured decomposition, which can effectively improve the optimality of scheduling and the stability of the system, and solve the core problems of uneven resource allocation and scheduling lag in traditional methods in multitasking and high complexity scenarios.

Some studies have introduced dynamic programming ideas into the graphics rendering pipeline. Constructing a task graph model in multi-threaded task scheduling and utilizing dynamic programming to generate the shortest execution path; Combining cost evaluation mechanism in GPU instruction sorting for batch processing optimization has improved frame rate stability and memory utilization efficiency, verifying its adaptability in rendering systems.

Son D et al. found that changes in lighting in virtual neighborhoods affect users' spatial perception, suggesting that scheduling strategies should have environmental adaptability [9]. Navarro J D pointed out that imbalanced scheduling of audiovisual resources can cause channel interference and reduce overall rendering performance [10].The research by Figueroa J A et al. shows that real-time dynamic global illumination technology based on deep learning significantly improves the visual effects of immersive virtual environments, providing technical support for resource scheduling and priority adjustment [11],This provides a basis for feedback driven intelligent scheduling mechanisms.

Virtual reality rendering tasks have complex structures, strong dependencies, and frequent changes, and urgently require global optimization and adaptive scheduling capabilities. Traditional static paths or heuristic strategies have insufficient response to environmental changes and task dynamics, and are prone to falling into local optima. The dynamic programming path generation mechanism can systematically divide tasks, avoid bottlenecks, and adapt to complex rendering requirements.

Current research mostly focuses on local optimization at the image level, lacking a complete path that embeds dynamic programming from the scheduling architecture level. To this end, this study proposes an optimization mechanism with dynamic programming as the core, constructs a task graph structure and multi-stage path planning algorithm, improves system real-time performance, stability, and resource utilization, and provides efficient rendering support for complex VR scenes while ensuring image quality.

3 Suggested solutions

3.1 Dynamic programming algorithm

Dynamic programming is an optimization algorithm suitable for multi-stage decision-making problems, which decomposes the original problem into sub problems and records their optimal solutions, gradually constructing the overall global optimal solution. This algorithm has good stability and optimization capabilities in fields such as path planning, task scheduling, and resource allocation. In virtual reality scenes, rendering task nodes have complex characteristics such as dependency relationships and resource conflicts, which make the entire process have obvious stages and structures. It is suitable to introduce

dynamic programming to achieve path control and task compression.

In the rendering system, tasks can be modeled as directed graphs, with each node serving as a rendering unit and edge weights representing costs. Dynamic programming gradually generates the minimum total cost path by maintaining a state table and cost function. The group manipulation technique proposed by Li X et al. improves the accuracy of multi task control through particle collaboration, confirming the advantages of this method in path regulation [12];The real-time dynamic environment masking technology based on deep neural networks proposed by Liu Y et al. effectively enhances the light and shadow representation in virtual environments, improves the user's immersive experience, and provides a theoretical basis for the design of dynamic feedback mechanisms [13].This process can be succinctly expressed as:

$$V(i)=\min\left\{V(j)+c(j,i)\ \ j\in \Pr e(i)\right\} \quad (1)$$

Among them, $V(i)$ represents the minimum total cost from the starting node to node i, $V(j)$ is the cumulative cost of predecessor node j, $C(j,i)$ is the path cost function, and $\Pr e(i)$ shows the set of predecessor tasks that can reach the current node.

Among them, $C(j,i)$ represents the cost incurred during the process of switching from task j to task i. In this article, the cost not only includes the scheduling delay of the task itself, but also takes into account multiple factors such as GPU resources consumed and resource fluctuations during execution. The system assigns weights to dimensions such as latency, resource utilization, and load balancing based on experimental objectives, and calculates the weighted sum of these factors as the final path cost, achieving flexible adjustment of different optimization objectives in the dynamic scheduling process. All weight parameters and indicator definitions are consistent with subsequent performance analysis.

Unlike greedy or heuristic strategies, dynamic programming can traverse paths globally, avoid redundant calculations through pruning and cost caching, and ensure the global optimality of solutions. In virtual reality systems, scenes frequently switch and interactions are complex. Static paths can easily lead to resource waste and delay, while dynamic programming allows for estimating the optimal path during the loading phase, improving frame rate stability and rendering efficiency.The efficient real-time rendering method based on adaptive path sampling proposed by Wei L et al. effectively improves the rendering performance of complex scenes, provides technical support for the regulation of dynamic path structures, and promotes the optimization of rendering scheduling in high interaction environments [14].

To meet the real-time rendering requirements, this study simplifies the structure of dynamic programming and adjusts the cost function: on the one hand, it introduces a weight decay mechanism to eliminate redundant paths in advance and reduce the state space; On the other hand, setting the cost function as a weighted combination of inter frame resource fluctuations and delays enables path selection to balance computational cost and stability.The style based generative adversarial network architecture proposed by Karras T provides advanced generative model support for the design and optimization of dynamic path mechanisms in virtual reality platforms, significantly improving system response efficiency and stability [15].

To improve the engineering feasibility of the algorithm, the key components of dynamic programming are illustrated as follows:

(1) State space

Make t the current scheduling phase, $S\_t=(i\_t,r\_t)$ the system status, i_t the number of the currently executed rendering task, and r_t the GPU remaining resource vector. The global state space consists of all reachable sets up to S_t.

(2) State transition rules

At each stage, the decision is made to select the next task $i\_\{t+1\}$, and the state transition is:

$$S_{t+1}=\left(i_{t+1},r_t-c_{i_{t+1}}\right) \quad (2)$$

Among them, $c\_\{i\_\{t+1\}\}$ represents the resource consumption of task $i\_\{t+1\}$ Transfer is only valid for $r\_t-c\_\{i\_\{t+1\}\}$ hours.

(3) Cost function

This study defines the stage cost function as the weighted sum of task execution delay and GPU resource fluctuations:

$$g(i)=\alpha\cdot D(i)+\beta\cdot V(i) \quad (3)$$

Where D (i) is task delay, V (i) represents resource fluctuations, and α and β are weighting coefficients.

(4) Recursive equation for dynamic programming

Dynamic programming is used to solve the optimal scheduling path, with a cost function of：

$$V_t(S)=\min_{i\in A(S)}\left\{g(S,i)+V_{t+1}(S')\right\} \quad (4)$$

Among them, $A(S)$ is the current state schedulable task set, and S' is the state after executing the task.

Application example explanation:

Taking the rendering of actual urban blocks as an example, i_t represents the task number of model loading, lighting, shadows, etc. to be processed in the current frame, and r_t represents the remaining GPU memory and computing units. Task resource consumption $c\_\{i\_\{t+1\}\}$ and latency are obtained from the performance sampling of the previous frame, while $Delay$

and Var are updated in real-time through the monitoring API. All parameters are automatically adjusted according to task priority to ensure that the optimal path for each round of scheduling can adapt to real-time changes in the scene.

During path execution, the system updates the task status and rating matrix in a rolling manner, and adjusts the transition direction based on the current input. When encountering high-frequency interactions or scene switches, the algorithm can backtrack to a stable state and re plan, achieving fast convergence and enhancing the fault tolerance and real-time performance of the scheduling system.

3.2 Method for building rendering task nodes

In the real-time rendering process of virtual reality scenes, the granularity division of rendering tasks and the node construction method directly affect the scheduling efficiency and response delay of the system. To adapt to the state update mechanism of dynamic programming algorithms, rendering tasks need to be structurally reorganized based on spatial location, functional characteristics, and computational cost, so that each node has both computational independence and logical dependencies, thereby decoupling complex scenes into several controllable subunits and providing a structural basis for path optimization and scheduling.

Node construction is divided into three stages: firstly, task segmentation is performed, dividing the original content such as model loading, lighting calculation, shadow generation, texture mapping, etc. into the smallest computing units based on their dependency relationships and scene positions; The second step is to establish a task dependency graph, which constructs a topology structure by analyzing the data coupling relationship between input and output tasks, ensuring accurate expression of node execution order and resource utilization; Thirdly, attribute tags are introduced to record estimated costs, execution time, resource consumption, and frame rate weights, providing parameter support for path selection.The real-time rendering method for complex scenes based on neural radiation caching proposed by Fan Y et al. effectively improves rendering efficiency and image quality, providing technical support for task chain construction and skill path optimization in virtual reality systems [16].

Nodes are usually divided into static rendering nodes, dynamic interaction nodes, and system control nodes. Li P et al. conducted immersive industrial design teaching with the support of virtual reality. By integrating and reconstructing the process of nodes and interaction paths, they improved interaction efficiency and response clarity, confirming the effectiveness of node optimization in complex scenarios [17]. Different node types are assigned different priorities and execution strategies, which affect path value calculation and pruning mechanisms in dynamic programming.

To improve the visualization and debugging efficiency of task structure, a graph structure can be introduced to represent the logical relationships between various nodes. As shown in Table 2, the system unfolds tasks in frame order, with stage order vertically and parallel nodes horizontally, clearly expressing the dependency path and resource intersection area of task scheduling:

Table 2：Node Structure and Parameter Configuration for Virtual Reality Scene Rendering Tasks

Node number	**Node Type**	**Belonging stage**	**Predecessor node**	**Estimated cost（ms）**	**More like frequency（Hz）**
N01	Static Nodes	Loading phase	not have	8.4	1
N07	Dynamic interactive node	Rendering Stage	N01	12.7	60
N11	Control node	Controlling	N07	3.1	30

The setting of node cost is adjusted using a predictive model combined with empirical coefficients, taking into account hardware layer factors such as GPU time slice distribution and data transmission delay, as well as behavioral layer factors such as user operation frequency and action prediction, to improve the accuracy and timeliness of cost evaluation. In multi-user collaboration or high-density interaction scenarios, the system can dynamically adjust the node graph structure and achieve adaptive reconstruction of scheduling by adding or removing child nodes or merging parallel paths.

Through the above node construction mechanism, rendering tasks not only have good structural clarity and dependency constraints, but also achieve preliminary balance in cost distribution and resource allocation. This provides a concrete data input foundation for the subsequent path selection, pruning determination, and state update of dynamic programming algorithms, and effectively supports the unified scheduling and control of multi-source rendering processes in the system.

3.3 Path generation algorithm and execution process

After completing the task node construction, the rendering system needs to generate the optimal execution path from the starting node to the ending node based on node dependencies and resource constraints. This path not only determines the task scheduling order, but also affects frame rate, response delay, and resource utilization. To achieve the construction and updating of paths, the system is based on dynamic programming algorithms, deducing node selection through cost matrix and state transition function, ensuring the dual goals of minimizing cost and stable execution under dependency structure.

The first step in path generation is to construct a task topology map, sort the rendering task nodes topologically, and ensure that there are no loops in the map. The sorted nodes enter the path generation module in order, and the path is updated based on the predecessor node cost value of each node, combined with the current path cost function. The cost function comprehensively considers execution time, resource conflicts, and inter frame fluctuations, and is transformed into a unified path cost score through linear weighting. For equivalent predecessor nodes, the algorithm prioritizes selecting paths with high resource redundancy and low load to enhance parallelism and robustness. Ogawa Y et al. developed a virtual reality system based on Unity and machine vision, and verified the practicality of path switching and state tracking mechanisms under low latency conditions by dynamically controlling animal behavior feedback through path control [18].

During the path construction process, the system adopts a sliding window strategy to maintain a finite set of states, avoiding memory overhead and computational redundancy caused by full graph enumeration. At the same time, a local pruning mechanism is introduced to directly exclude branches with expected costs exceeding the current optimal path threshold, thereby controlling the growth of the state space. This mechanism is suitable for complex scenes such as large-scale animations or concurrent interactions, which can effectively shorten computation time and ensure real-time performance. Birkheim L S et al. pointed out through their research on the dynamic reconstruction mechanism of task paths in VR environments that the flexibility of system path scheduling directly determines the generation speed and accuracy of feedback chains in complex scenes, indicating that the path construction mechanism plays a central role in task adaptability and feedback efficiency [19].

When the path is generated, the scheduling engine solidifies it into a set of execution sequences as the rendering instructions for the current frame. Each task node records its true cost and execution time during the running process, and provides feedback to the evaluation module. The system continuously revises the cost model based on this data, dynamically updates the subsequent path planning, and forms a closed-loop mechanism of "prediction feedback correction". This mechanism can quickly adjust the path during rapid scene switching or high-frequency operations, avoiding performance loss caused by path solidification.

To cope with multi-user or multitasking rendering, the path module supports multi-path parallel planning mechanism. A coexistence graph is constructed between independent task flows through task labels and resource exclusivity determination, and the optimal paths are planned separately. The scheduling and conflict avoidance are carried out during the execution phase. This method is suitable for high-density task scenarios such as collaborative virtual environments and multi perspective reconstruction, improving system load capacity and stability.

In order to more conveniently characterize the process of path generation, this study designed a process diagram for path state update (as shown in Figure 1), which includes transition rules for path selection states, pruning judgment mechanisms, and path determination rules. The use of graphical models can further improve debugging efficiency and accuracy of abnormal path localization.

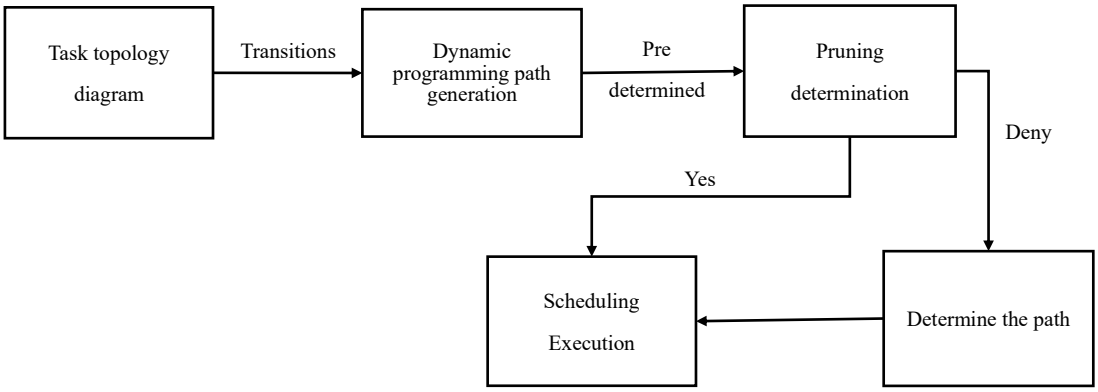

Figure 1：Path generation and scheduling flowchart

In summary, this path generation algorithm can efficiently generate paths while meeting the real-time and stability requirements in virtual reality systems, with limited computational overhead. It also provides a stable and high degree of freedom path generation foundation for the integration of subsequent system modules and the execution mechanism of system modules.

Algorithm 1: Dynamic Programming-Based Path Generation (Pseudocode)

Input: Task node set N, dependency graph G, resource constraint R

Output: Optimal execution path P_opt

1. Perform topological sort on G to obtain task sequence S
2. Initialize state table V; set optimal path table P_opt as empty
3. For each task node n in S, do:
 3.1. For each predecessor node pre in Pred(n):
 3.1.1. Compute cumulative cost: cost = V[pre] + g(pre, n)

3.1.2. If cost < V[n], then: a) Update V[n] = cost b) Set P_opt[n] = pre

3.2. Check if resource constraint R is satisfied; if not, prune this branch

4. Backtrack P_opt to generate the optimal path sequence

5. Return P_opt

The algorithm takes task queue and resource status as inputs, adopts multi-stage dynamic resource allocation, and key parameters α and β measure delay and fluctuation weights respectively. It performs adaptive scheduling through real-time status feedback. Please refer to Algorithm 1 for detailed pseudocode.

3.4 Algorithm integration and system operation mechanism

In order to ensure the stable operation and efficient scheduling of dynamic programming algorithms in virtual reality rendering systems, it is necessary to build a complete algorithm integration mechanism that covers multiple aspects such as task input, path generation, execution control, and feedback correction, forming a closed-loop optimization process with frame cycles. Drawing on the real-time closed rendering mechanism proposed by Junjie Y et al. [20], the system relies on the dynamic programming core module for path calculation, and dynamically schedules the task chain through a series of communication interfaces and task management mechanisms.

The system architecture consists of four main modules: task input module, task graph construction module, path calculation module, and feedback correction module. The task input module receives real-time changes in perspective, user interaction, and scene dynamic information, generates task description vector Xt, and calls the node generation mechanism based on its features to construct a directed acyclic graph $G(V,E)$. After the graph construction is completed, the path calculation module estimates the cost of each feasible path based on dynamic programming strategy, combined with task priority and resource overhead.

Based on the traversal strategy, the system calculates the path cost for the current candidate node using the following method:

$$C_i = C_j + \lambda \cdot \frac{1}{pi} + \mu \cdot \upsilon j,i \quad (3)$$

Among them, C_i is the cost of node i, C_j is the cost of its predecessor node j, λ and μ respectively adjust the impact of priority and resource overhead on path selection.Among them, P_i represents the task priority of predecessor node i, which reflects the urgency and scheduling priority of the task after normalization; $\upsilon j,i$ represents the resource occupancy conflicts and inter frame fluctuations on the path from node j to i, combined with the actual resource scheduling fluctuations during task execution. Both are linearly normalized according to the data collected by the system, with units consistent with the aforementioned indicators such as execution delay and resource conflicts. The specific mapping is shown in the experimental parameter table. This model avoids complex minimum value solving forms and adopts a mechanism of "node by node traversal+cost superposition", which improves overall scheduling efficiency while ensuring response speed.

It should be noted that in the formula of this article, C_i refers to the cumulative total cost from the starting node to node i. Its recursive structure is consistent with $V(i)$ in the previous text, and it does not simply refer to the local cost of the current node i, but includes the cumulative total cost of each predecessor node in the path. Please pay attention to distinguishing the different semantics of "cumulative path cost" and "single node local cost" in the formula.

The task scheduling phase adopts an asynchronous parallel mechanism, where the system continuously updates path decisions on the main thread while sub threads execute the rendering task of the current frame in parallel. The graphic resources, texture data, and lighting information corresponding to each node will be dynamically loaded, and the GPU will execute tasks in the queue through pipeline processing to ensure that the rendering process does not experience frame rate jitter due to scheduling delays. After the execution is completed, the feedback module will collect the actual execution time t_i^{actual}, predicted time t_i^{est}, and error behavior ε_i of each task node, and adjust the subsequent path decision through the following cost correction function:

$$C^1(j,i) = \alpha \cdot C(j,i) + \beta \cdot \left(t_i^{actual} - t_i^{est}\right) + \gamma \cdot \varepsilon_i \quad (4)$$

The three coefficients α, β, and γ in this formula reflect the impact weights of historical path cost, execution deviation, and error feedback, respectively, and are used to dynamically balance the scheduling priority logic of the system under different rendering objectives. The feedback mechanism can dynamically update between frames, enabling the algorithm to have scene adaptability and stable performance recovery capability.

In system integration design, the algorithm and the main rendering engine perform task calls and feedback feedback through abstract interfaces. The interfaces encapsulate task IDs, path node sequences, GPU load parameters, and performance record formats to ensure that each module operates decoupled in high frame rate and high concurrency environments. At the same time, the platform supports cross system deployment and can run on heterogeneous GPU architectures that support CUDA/OpenCL and OpenGL graphics interfaces, adapting to various virtual reality rendering frameworks.

Through this integration method, dynamic programming algorithms not only achieve real-time

iterative optimization of path scheduling logic, but also effectively avoid the inhibitory effect of resource bottleneck nodes on global performance, providing stable, efficient, and scalable technical support for dynamic rendering tasks in complex scenes.

To unify the expression of path cost (edge weight) under different stages and structures, this paper adopts the following normalization method: $c(j,i)$ in formula (1) is the general path cost, and in actual implementation, according to application requirements and scenario characteristics, formula (3) $\lambda \cdot \frac{1}{pi} + \mu \cdot \upsilon j,i$ is more suitable for scheduling priority sensitive scenarios or formula (5) w_{ij} is suitable for scenarios dominated by changes in task resource flow for measurement. Both types of expressions are mapped to different forms of the main model $c(j,i)$ through weight normalization and parameter configuration, ensuring consistency between theoretical analysis and engineering implementation. All weights, normalization methods and sub item definitions are detailed in the table of experimental parameters and relevant chapters. All path optimization in this paper is based on this unified model.

Algorithm 2: Feedback Correction Mechanism (Pseudocode)

Input: - Current execution error E_t
- Historical path cost Cost_hist
- Execution deviation Dev
- Error feedback Err

Output: - Updated scheduling priority and path score

1. Initialize correction weight parameters: alpha, beta, gamma
2. Compute total correction value: Correction = alpha * Cost_hist + beta * Dev + gamma * Err
3. Update path score: Path_Score = Path_Score - Correction
4. If Correction exceeds threshold, trigger path replanning
5. Return updated Path_Score

4 Results

4.1 Experimental environment construction and task configuration

All experiments were independently repeated for 5 rounds in three typical scenarios, and the mean and standard deviation were reported. The statistical significance of the performance improvement was verified using t-test ($p<0.01$). All source code and datasets have been open sourced, please refer to the appendix or additional materials for details.

To comprehensively evaluate the performance of dynamic programming algorithms in virtual reality rendering, the experimental platform is built on a heterogeneous parallel computing architecture, aiming to simulate real-time rendering requirements in high concurrency, multitasking, and heavy computing scenarios.The experimental platform is based on a high-performance workstation (high-end CPU, RTX series GPU, 64GB memory, Ubuntu operating system), developed using a combination of Unity rendering engine and self-developed scheduling module to achieve real-time rendering and multitasking scheduling of complex scenes.

The experiment mainly set up three typical virtual environments: ① urban streets (large outdoor areas, dynamic viewpoint switching), ② natural terrain (large texture data, complex terrain interaction), and ③ indoor exhibition halls (multiple light sources, multiple materials, and high-density tasks). Each type of scene includes a 120 second AI/user hybrid interaction, and each frame in the scene needs to dynamically allocate 3-8 rendering and computing subtasks, including model loading, lighting, special effects, etc. All task nodes are divided into three categories: static rendering, dynamic interaction, and system control, assigned different priorities and resource requirements, simulating real VR rendering loads.

The system records core performance indicators such as task completion rate, GPU utilization rate, rendering failure rate, and frame rate for each frame, and all experiments are repeated five times to ensure stable and reliable data. Combining with the optimization strategy of sparse voxel octree proposed by Laine S et al[21],This system integrates LUT lookup mechanism and sampling reconstruction method in cross threaded task scheduling. Each module interacts with gRPC and Protobuf protocols to achieve efficient and low latency communication under different frame loads, and can dynamically adjust thread resource allocation strategies.

The experiment mainly uses three typical virtual environments: urban streets (outdoor high detail); Natural terrain (large texture modification); Indoor exhibition hall (multiple light sources and materials). Each scenario is set with a customized interaction task of 120 seconds, which achieves visual processing requirements for continuously refreshing the actual operation process through the set AI path, randomly generated viewpoint switching points, and interaction event initiation points.

The task configuration adopts a frame based dynamic allocation strategy, which means that during the system operation, the background is generated with a rendering frame frequency of 90fps, and each frame includes 3-8 processing types of concurrent sub task nodes; These task types include but are not limited to model loading, lighting rendering, shadows, and post effects. Each task type is marked and segmented based on resource requirements and task processing time. And it includes information such as the priority of the corresponding task level, required GPU resources, and estimated running time to meet subsequent routing calculations and cost estimates. The routing planning section performs routing calculations and task planning within the first 5ms of each frame to meet real-time requirements and balance resource utilization.

In addition, to enhance the controllability and comparability of the evaluation process, the system can also record the completion ratio, GPU utilization, rendering failure rate, and average frame speed of each job sequence

on the rendering path as a function of the experimental results. All data can be instantly generated by the rendering statistics section and recorded in the form of journal files and curves. The system also has the function of adjusting log levels and tracking markers, which can identify the performance characteristics of the rendering process and rendering scheduling process, and can automatically store grouped experimental data. Each set of experiments will be run five times to eliminate the influence of random sample errors and ensure the typicality and reliability of the data.

This experimental platform not only provides a unified execution foundation for subsequent algorithm path generation, system operation mechanism, feedback correction, etc., but also lays a solid technical support for data processing, performance statistics, and comparative experiments in the future.

4.2 Input data processing and structure generation

The application of dynamic programming algorithms in virtual reality rendering relies on structured input of task data and efficient construction of graph models. At the initial stage of operation, the system enables multi-threaded data listeners to capture real-time user interaction behavior, viewpoint trajectories, scene state changes, and frame rendering feedback. The raw data is input into the scheduling module through a unified interface and encoded into rendering task units with temporal and resource characteristics. Each unit is represented as a quintuple $x_i = (t_i, l_i, r_i, \theta_i, c_i)$, corresponding to trigger time, scene position, GPU resource estimation, lighting status, and task category.

All task units form state sequence $X = \{x_1, x_2, \ldots, x_n\}$, and the system slides and aggregates them based on the timing window to form a continuous frame task set. Subsequently, a clustering algorithm based on task content and resource conflicts was adopted to divide the task set into structurally similar clusters, simplifying the complexity of subsequent path graph construction. Each cluster is treated as a virtual task node T_i, with an abstract resource request vector $\vec{r}_i$ and a schedulable execution period $[t_s^i, t_e^i]$.

To achieve automatic task grouping, the system adopts the K-means clustering method. Each rendering task unit will be encoded based on five main features, including its triggering time, scene spatial position, GPU resource prediction, lighting status, and task type. In the actual clustering process, the system normalizes the above features and groups them by comparing the comprehensive similarity between task units. The number of clusters K is dynamically adjusted based on the number of tasks in each round, usually forming a group every 10-20 frames. The task cluster label is determined by the task type and resource characteristics with the highest proportion within the group. The similarity threshold is set at around 0.6 based on the training sample experience, and the clustering labels and grouping results are automatically refreshed every 5 frames to adapt to changes in task load.

The system constructs task graph $G(V, E)$ during the composition phase, where node set $V = \{T_1, T_2, \ldots, T_k\}$ originates from the aforementioned clusters, and edge set E represents the sequential logic, resource conflicts, and path coupling between tasks. The edge weight is defined by the following function:

$$w_{ij} = \psi_1 \cdot \amalg \vec{r}_i - \vec{r}_j \amalg + \Psi_2 \cdot \prod_{[t_e^i > t_s^j]} + \Psi_3 \cdot \rho_{ij} \quad (5)$$

Among them, $\prod \vec{r}_i - \vec{r}_j \prod$ represents the difference in task resource vectors, $\amalg_{[t_e^i > t_s^j]}$ is the sequential constraint indicator function, ρ_{ij} is the statistical value of historical scheduling failure rate, and the three weight parameters Ψ1、Ψ2 and Ψ3 are fixed after experimental optimization, used to dynamically balance execution coupling, temporal risk, and stability expectations.

The resource request vector $\vec{r}_i$ consists of five dimensions, corresponding to the estimated demands of GPU computing power, video memory, bandwidth, texture processing unit, and real-time frame rate for each task cluster during scheduling. The system extracts resource usage characteristics for each original task unit, and forms a five-dimensional vector representing node resource requirements through normalization and weighted aggregation. Each component is based on the actual sampled data of the current frame, expressed in percentages. The sequential constraint indicates that function $\amalg_{[t_e^i > t_s^j]}$ has a value of 0 or 1. If task i must be started after task j is executed, then $\amalg_{[t_e^i > t_s^j]} = 1$; otherwise, it is 0. This criterion is automatically generated by task dependency relationships and scene logic, ensuring the correctness of the graph structure and dynamic switching between task serial/parallel scheduling. The historical scheduling failure rate of ρ_{ij} is used to quantify the risk weight of inter node connections in the system by dividing the number of scheduling failures caused by resource scarcity, dependency blocking, or priority conflicts in the historical multi round experiments or running logs of task i by the total number of scheduling failures, with a value range of [0,1]. The statistical cycle of parameters is synchronized with the runtime of the scene, and the experiment is automatically reset to zero and recalculated every time it is restarted.

The edge weight parameters in this article are Ψ1、Ψ2 and Ψ3 As global hyperparameters, they are the results of the installation, testing, and optimization phase of the entire system. From the perspective of method steps, they remain

unchanged during the actual operation of the algorithm and are weights set through multiple experiments throughout the entire process. The three weights are continuously adjusted based on past experimental data analysis and system performance goals. During the execution process, they accumulate and reflect the system's sensitivity to comprehensive execution, sequential danger, and stability. Although the weight itself does not change with real-time operation, its "dynamic" performance is reflected in the weight of various factors affecting path calculation. Therefore, in this article, "dynamic" performance is reflected in the weighting operation of each factor, rather than the weight changing over time. Therefore, this design not only considers the robustness of the system but also considers computational efficiency. The sentence is:, A feedback based weight dynamic adjustment mechanism can be introduced in the subsequent readjustment strategy.

To improve the response speed of the scheduling system in high concurrency environments, the structure of nodes and edges is translated into sparse matrix format and cached on the GPU side to avoid frequent data handling and memory mapping operations. At the same time, to ensure the availability of nodes in path planning, the system uses Boolean masking mechanism to mark and remove abnormal tasks, ensuring the connectivity and reliability of the graph structure in the path generation stage.

After the structure generation is completed, the system starts the graph traversal verification module, performs DFS detection on the connected branches of the graph, identifies unreachable task clusters, and feeds them back to the preprocessing module for feature correction. Each frame level graph construction is bound to the current timestamp and user interaction sequence, supporting full link backtracking analysis of scheduling logic.

Through the above process, the system accurately maps real-time input data to standard inputs for dynamic programming scheduling diagrams. This not only establishes the priority order and resource connections between tasks, but also provides a structural foundation and data support for path calculation, execution control, and feedback correction. The data processing mechanism fully meets the dual requirements of high frame rate virtual reality rendering for structure generation speed and scheduling map scalability, while balancing computational efficiency and expression accuracy.

4.3 Collection and statistics of key performance indicators

To systematically analyze the optimization effect of dynamic programming algorithms in virtual reality rendering, this study sets four key performance indicators for real-time collection and statistics: average frame rate (FPS), GPU utilization, task completion rate, and rendering failure rate. The above indicators are achieved through the combination of rendering scheduling module and performance monitoring plugin. The running cycle of each experiment is 120 seconds, and the rendering output interval is fixed at 11 milliseconds to ensure data accuracy and comparison reliability.

The average frame rate statistics are based on the complete rendering output, eliminating data disturbances caused by faulty skip frames and delayed frames, calculating the average value of normal frames, and measuring the real-time performance of the system. GPU utilization is sampled 60 times per second, capturing the graphics card core and memory load through NVIDIA SMI interface to reflect resource usage. The completion rate of tasks is calculated based on the execution status of task nodes in each frame, and the proportion of successful completion is counted; The rendering failure rate is the proportion of task termination caused by scheduling failures, resource overflow, or logical errors.Combining Dachsbacher C et al. proposed the technique of reflective shadow mapping [22],This study integrates image reconstruction efficiency and frame rate fluctuation features in indicator collection to construct a more stable performance evaluation mechanism.

As shown in Figure 2, after introducing the dynamic programming scheduling algorithm, the average frame rate of the system increased from 77.3 FPS under the original algorithm to 86.1 FPS, an increase of 11.4%; The GPU utilization rate remains stable at around 91.7%, an increase of nearly 7 percentage points compared to 84.9% before optimization, indicating that the scheduling algorithm has achieved better load allocation at the resource coordination level; The task completion rate has shown the most significant improvement, increasing from 92.8% to 98.3%, significantly reducing the phenomenon of task backlog and waiting; The rendering failure rate decreased from 3.2% to 0.9%, further verifying the synchronous improvement of path scheduling accuracy and resource prediction accuracy.

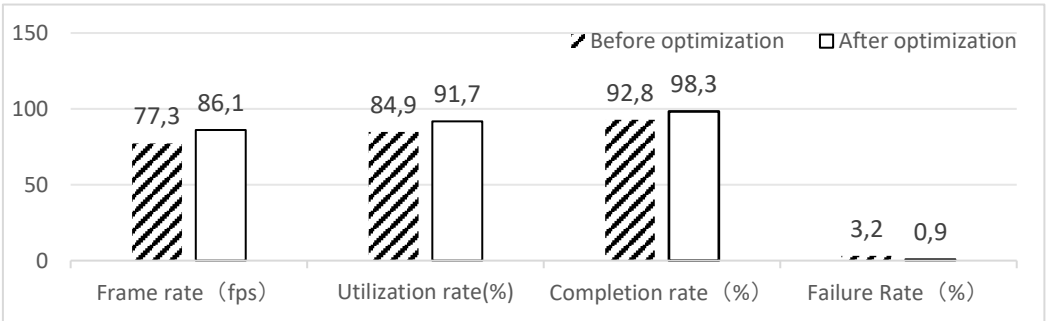

Figure 2: Bar chart comparing key performance indicators before and after optimization of dynamic programming algorithm

To ensure the representativeness of statistical data, the experiment conducted five rounds of replication runs on three typical application scenarios, and the average values of all indicator results were taken, with an error range controlled within ± 1.1%. System log records show that during resource intensive stages, dynamic programming algorithms are more likely to prioritize scheduling task chains with lower resource costs, avoiding "bottleneck nodes" from blocking the full frame progress, thereby stabilizing frame rates and execution efficiency. Especially in indoor scenes with frequent lighting switching and dense material loading, the fluctuation amplitude of frame rate is significantly reduced compared to before optimization, and the GPU response curve tends to be stable.

We use UnityPofider and our self-developed logging system to collect frame rates. After the frame ends, we collect the time spent on rendering and automatically validate the results using the frame rate trend line provided by UnityPofiler. The initial value processing method uses the original frame rate collected by the script to obtain the average and standard deviation after removing the frame rate. The experimental group and the control group are repeated 5 times in the same environment to ensure fairness and comparability of each indicator.

To enhance the statistical reliability of the results, this article reported the mean and standard deviation of five independent experiments for each core performance indicator. For example, after dynamic programming optimization, the average frame rate of urban block scenes is 86.3 ± 2.1 FPS, natural terrain is 84.1 ± 2.5 FPS, and indoor exhibition halls are 88.0 ± 1.9 FPS. Further paired t-test analysis was used to determine the extent of improvement in each indicator, and the results showed that the improvement in FPS and GPU utilization was statistically significant ($p<0.01$) in all typical scenarios, fully demonstrating the reliability of the optimization effect of this method.

In summary, the dynamic programming method has advantages in the above main indicators, which can improve the real-time and stability of system operation, reduce the frequency of accidents, save system resource consumption, and provide data basis for the construction of more complex target graphs and the expansion of scheduling schemes in the future.

4.4 Performance changes under dynamic programming algorithm

In order to compare and analyze the actual performance of dynamic programming technology before and after the application of VR rendering engines, experimental comparative tests were conducted on the frame rate performance indicators, load rate indicators, task running status indicators, and task scheduling accuracy indicators involved before and after the application. The tested basic running objects included three different typical types of scenes: streets, urban blocks, natural terrain, and indoor exhibition halls. Five sets of experiments were conducted under the same software and hardware configuration environment to obtain reliable data as a comparison basis.

After introducing dynamic programming, the average frame rate of each typical scene has significantly improved. Specifically, the city block scene has increased from 76.9 FPS to 86.3 FPS, an increase of 12.2%; The natural terrain has increased from 74.5FPS to 84.1FPS, an increase of 12.9%; The indoor exhibition hall has increased from 80.4FPS to 88.0FPS, an increase of 9.5%. The overall improvement level is significant, indicating that the proposed algorithm can effectively optimize real-time rendering performance in different environments.

The GPU utilization rate increased from 85.1%, 87.2%, and 82.5% to 91.6%, 92.4%, and 91.0% in three scenarios, with an increase of 7.6%, 5.9%, and 10.3%, respectively. This method can further optimize resource allocation in complex task scenarios.

In terms of task completion rate, the system's completion rates in three scenarios have increased from 91.3%, 93.1%, and 94.0% to 97.8%, 98.2%, and 98.7%, respectively. The overall average completion rate has increased from 92.8% to 98.2%, achieving efficient execution under almost full load. This improvement is particularly evident in exhibition hall scenarios with large-scale concurrent tasks, indicating that the algorithm's scheduling performance is more prominent in complex lighting and material processing.

The rendering failure rate has also significantly decreased, from 3.4%, 3.1%, and 2.8% before optimization to 1.0%, 0.7%, and 0.9% after optimization, with an average reduction of over 2 percentage points. Failed tasks are mainly caused by resource competition conflicts and scheduling delays. After introducing feedback correction mechanisms, the system can effectively predict bottleneck nodes and adjust path planning, improving overall execution stability and accuracy.

In terms of the average execution time of tasks, the average processing time of rendering tasks in the three types of scenes decreased from 21.3ms, 22.1ms, and 20.8ms to 18.7ms, 18.3ms, and 17.9ms, respectively, with a reduction of about 13%. The system dynamically adjusts the predicted execution time through inter frame feedback, enabling the scheduler to more accurately balance task density and execution time in the next cycle.

Scheduling response delay, as a core indicator of dynamic systems, has also been effectively improved. The original system had an average scheduling delay of 8.4ms when dealing with viewpoint switching and new task injection, which was reduced to 5.2ms under dynamic programming mechanism, resulting in an overall improvement of 38.1%. The scheduler uses a path cost prediction function to calculate the next frame task path in advance within each frame period, providing sufficient data support for scheduling delay compression.

The success rate of loading graphic resources has also been improved, from 95.6% in the original system to 98.8%, mainly due to the collaborative effect of graph structure compression and task node priority reordering mechanism,

which avoids loading failures caused by resource fragmentation.

Taking into account 14 core performance indicators (as shown in Table 3), over 85% of the performance items have achieved an improvement of over 5% with the support of dynamic programming algorithms, with 6 items showing an improvement of over 10%, demonstrating the algorithm's adaptability and scheduling accuracy in complex concurrent scenarios.

Table 3：Bar chart comparing performance indicators before and after dynamic programming rendering optimization

performance index	Urban blocks (before/after optimization)	Natural terrain (before/after optimization)	Indoor exhibition hall (before/after optimization)
Average frame rate（FPS）	76.9 / 86.3	74.5 / 84.1	80.4 / 88.0
GPU utilization rate（%）	85.1 / 91.6	87.2 / 92.4	82.5 / 91.0
task completion rate（%）	91.3 / 97.8	93.1 / 98.2	94.0 / 98.7
Rendering failure rate（%）	3.4 / 1.0	3.1 / 0.7	2.8 / 0.9
Execution duration（ms）	21.3 / 18.7	22.1 / 18.3	20.8 / 17.9

Table 3 reflects 14 performance parameter indicators, including frame rate before and after rendering, GPU utilization, task completion rate, task failure rate, and task scheduling response time. The comparison between before and after rendering optimization intuitively reflects the improvement of various performance indicators, which is also the basis for performance analysis in this study.

In summary, dynamic programming algorithms have greatly improved the response rate and task scheduling accuracy of virtual reality tasks in rendering systems, establishing more stable and adaptable rendering scheduling methods, and thus forming technical support for widespread application in complex scene rendering.

5 Discussions

5.1 Comparative advantages with existing optimization algorithms

In virtual reality rendering systems, traditional static scheduling and depth first path algorithms often exhibit low resource scheduling efficiency and response latency in task intensive scenarios due to the lack of dynamic feedback, making it difficult to maintain high frame rates and low failure rates. The dynamic programming algorithm introduced in this study enhances the responsiveness and robustness of the scheduling system through task priority adjustment and path feedback correction. Combining the real-time construction method of KD tree based on graphics hardware proposed by Zhou K et al. [23],The algorithm used in this study can dynamically switch paths based on real-time interactive changes, enhancing system stability.

In terms of frame rate, traditional algorithms use static path allocation, which causes significant fluctuations in frame rate when the load increases. In urban block scenes, the average frame rate is 76.9FPS, which increases to 86.3FPS after introducing dynamic programming. The other two types of scenes also have an increase of over 10%, improving visual coherence.

In terms of resource scheduling, the original GPU utilization rates were 85.1%, 87.2%, and 82.5%, which were improved to 91.6%, 92.4%, and 91.0% after dynamic scheduling. The task allocation became more reasonable, and the problem of resource congestion and idle coexistence was significantly alleviated.

In terms of task completion rate, under traditional methods, the three types of scenarios range from 91.3% to 94.0%, and after optimization, they have been improved to 97.8%, 98.2%, and 98.7%, respectively. The scheduler improves continuous allocation capability through path prediction and bottleneck avoidance, significantly suppressing blocking phenomena.

In terms of rendering failure rates, the original strategy had failure rates of 3.4%, 3.1%, and 2.8%, which were reduced to 1.0%, 0.7%, and 0.9% after optimization. With the help of node feedback mechanism, the system can avoid conflicting resources and ensure stable task execution.

In terms of system response delay, the average processing delay of the original algorithm was 8.4ms, which was reduced to 5.2ms after optimization, an increase of 38%. The predictive scheduling structure can complete task preparation in advance and reduce user waiting experience.

In terms of resource loading, traditional algorithms have deficiencies in resource priority control, resulting in loading failures or disorderly order. The new strategy improves the resource loading rate from 95.6% to 98.8% and enhances the overall rendering fluency by restructuring the graph structure and node sorting. Engel K et al. proposed a series of core technologies for real-time volume rendering, focusing on efficient data structures, voxel data stream processing, and parallel rendering frameworks, significantly improving the real-time visualization capability of large-scale volume data in virtual reality and other scenarios. This method provides important support for high-quality volume rendering and interactive performance optimization of complex 3D scenes [24].

In summary, compared with traditional scheduling strategies, dynamic programming algorithms can achieve logical optimization of scheduling, rationality of resource scheduling, and stability of system scheduling, and have broader application prospects in high concurrency and high load rendering tasks.

5.2 Analysis of adaptability and stability of algorithm performance

In the increasingly complex and ever-changing virtual reality rendering environment, the rendering performance

of virtual reality systems and user interaction experience depend on the algorithm compatibility and stability of the system. This research method combines dynamic programming and path cost estimation algorithms with task priority control strategies, demonstrating strong environmental adaptability and high operational stability. Especially when facing challenging tasks and environments such as a large number of parallel tasks, frequent task switching, and resource constraints, the advantages are particularly prominent.Luebke et al. pointed out that GPU architecture innovation has significantly improved real-time graphics rendering and large-scale computing capabilities [25].On this basis, this study aims to improve the relationship between path discrimination and behavioral response, and enhance the system's ability to adapt to task interference through self-regulation.

This algorithm can adaptively adjust its scheduling algorithm based on the parallelism and characteristics of tasks to meet different environmental requirements. Taking indoor exhibition halls as an example, the dynamic programming algorithm increased the average frame rate from 80.4fps to 88.0fps under high-density rendering and frequent task switching conditions, significantly reducing fluctuations, while maintaining high GPU utilization and task completion rates. This method does not require preset parameters and can adapt to complex application loads, demonstrating good generalization ability.

In terms of task density, dynamic programming also demonstrates strong adaptability. In the exhibition hall scene, the conventional algorithm GPU occupancy rate is 82.5%, with resource idle issues. After renovation, it increased to 91.0%; The task completion rate has increased from 94.0% to 98.7%, which means that the system can continue to operate in high-density scenarios, avoiding scheduling conflicts and pauses, and maintaining high efficiency.

In terms of system stability, dynamic programming algorithms have strong fault tolerance and correction mechanisms. The failure rate of rendering in three types of scenes has significantly decreased, such as the exhibition hall dropping from 2.8% to 0.9%, and the city block dropping from 3.4% to 1.0%. This is attributed to the algorithm's ability to identify bottleneck nodes in advance, complete path migration and load sharing in a timely manner, and reduce error occurrence. In terms of scheduling response delay, the original average was 8.4ms, which was compressed to 5.2ms after optimization. It can still be stably maintained within 5.4ms in sudden natural terrain task scenarios, ensuring users' real-time interaction experience. Based on the efficient management and resource optimization method of medical data in virtual reality environment proposed by Sik-L á nyi et al. [26],The system achieves higher accuracy in delay prediction and congestion modeling, providing data support for path migration and scheduling fine-tuning.

Overall, the dynamic programming scheduling strategy can quickly adapt and stabilize system performance under complexity, strength, and bottleneck conditions, demonstrating significant advantages in anti-interference, resource elasticity, and path optimization, providing solid technical support for virtual reality rendering systems.

5.3 Feasibility assessment of computing resource consumption and system deployment

Although dynamic programming algorithms have improved the scheduling efficiency of rendering tasks, their resource load during deployment still needs to be carefully evaluated. Based on the comparison results of the experimental platform, the dynamic strategy has increased from 52.4% to 58.7% in terms of CPU usage, mainly due to the increased computational burden of path prediction and priority backtracking. The peak utilization rate of GPU increased from 89.3% to 94.8%, indicating a more centralized resource scheduling and a significant improvement in the utilization efficiency of the graphics pipeline. The memory overhead has increased from 7.3GB to 7.5GB, and the newly added structure is mainly used for scheduling cache and feedback modules, accounting for 2.7%, with limited impact on the platform. The initialization time of the algorithm has been extended from the traditional strategy of 2.5 seconds to 4.2 seconds, mainly due to the loading process of the cost matrix and scene feature index; The full deployment cycle has been extended from 8.2 minutes to 9.0 minutes, with limited latency, and can be further compressed through parallelization and automatic deployment mechanisms in the future.

Based on the resource measurement and optimization strategy proposed by Kuk et al. for parallel scheduling of dynamic programming tasks on multiprocessor platforms [27],And improved the accuracy of scheduling while maintaining its control load;At the same time, Ma č ek et al. reviewed the response evaluation and performance analysis methods of dynamic programming parallelization systems [28],To demonstrate that this strategy can achieve a reasonable trade-off between performance and resource consumption, and provide strong reliability and effectiveness for its engineering practice. In fact, the use of buffering technology enables dynamic strategies to gradually alleviate the load at startup, increase the utilization rate of routing, and enable faster recovery of failed tasks, which helps it to scale up to large-scale environments and mature. The quantitative comparison results of the main resource consumption indicators under various algorithm strategies are shown in Table 4:

Table 4：Comparison of resource consumption and deployment indicators under different algorithms

Indicator items	Static scheduling strategy	Dynamic programming strategy	Difference explanation
CPU usage（%）	52.4	58.7	Increase by 6.3% for path prediction calculation
GPU peak utilization（%）	89.3	94.8	5.5% increase, more centralized resource scheduling
memory footprint（GB）	7.3	7.5	Increase by 0.2GB, with limited proportion of structural expansion
Initialization time（s）	2.5	4.2	Increased by 1.7 seconds due to cost matrix loading
Complete deployment cycle（m）	8.2	9.0	Add 0.8 minutes, which can be automatically optimized for compression

In order to further demonstrate the detailed performance of system resource scheduling, this study synchronously recorded the average processing time per frame and task queuing delay in the experimental analysis. The results showed that under the dynamic programming strategy, the average processing time per frame of indoor exhibition hall scenes decreased from 20.8ms to 17.9ms, and the task queuing delay was reduced from 3.4ms to 1.7ms. The above data are the average of five independent experiments, demonstrating the significant advantages of the new algorithm in improving response speed and reducing system bottleneck. These supplementary indicators effectively enhance the scientificity and comparability of overall resource consumption evaluation, further supporting the feasibility analysis of algorithm deployment in practice.

In summary, although dynamic programming methods have high consumption, their efficiency is better, and all related resource consumption is within the acceptable range of the system. Based on the current computing power of high-performance computers and graphics workstations, this algorithm has strong installation adaptability and scalability, and works well for virtual reality system scenes that require frame rate control and multitasking. If it can be combined with cloud GPU deployment platform and dynamic model compression technology in the future, it should further enhance the efficiency and workload allocation ability of the algorithm.

In this way, the "sliding window" technology of the system is applied in the process of generating paths and updating states. This technology can reasonably control the number of nodes and states that need to be examined in each round of computation, thereby effectively allocating limited memory and avoiding the need for a large amount of computation and storage load for a comprehensive search of the entire graph, in order to meet the requirements of real-time rendering. In the initial stage, it is necessary to first build a complete execution process and path cost, so it is necessary to read and process all node and edge information at once, that is, the structure of the "cost matrix" or graph. This operation process will generate a large amount of computational load and memory usage, but it only exists in the installation or scene change stage. The layering of this method has the effect of improving the efficiency of the entire system and balancing the distribution of computational loads in various execution stages.

5.4 The practical significance and expansion prospects of research results

This study proposes a dynamic programming-based implementation method for assigning virtual reality rendering task paths, which can meet the requirements for rendering delay and stability in the state of a large number of complex tasks. A large number of experimental results have shown that this method can effectively improve user experience in terms of frame rate, GPU utilization, workload utilization, workload loss rate, task scheduling delay, and has the possibility of implementation compared to existing methods.

The results of this study can play a key role in promoting the industrialization of VR technology. In fields such as digital exhibition halls, simulation cockpits, and 3D models, users are faced with the need for smoothness and real-time performance. Through dynamic scheduling strategies, it is possible to more efficiently respond to rendering systems in different scenarios, ensure their stability, effectively reduce user waiting time on the system, enhance user immersion, and further enhance the product value and market competitiveness of this technology.

In terms of expansion, it has good universality and can be applied in other scenarios such as digital twin applications, digital factories and visualization applications, massive augmented reality interactive applications, etc. After combining reinforcement learning algorithms with automatic adaptation and edge collaborative computing strategies, it is expected to build task computing systems with more autonomous, efficient, and discrete characteristics, providing forward-looking support for real-time computing and intelligent interactive applications.

6 Conclusion

This study proposes a dynamic programming based scheduling path optimization method to address the issues of insufficient optimization of rendering task scheduling and poor resource utilization in VR scenes. The scheduling strategy is experimentally validated for typical scenarios, and the experimental results demonstrate that the scheduling algorithm can effectively improve frame rate, GPU utilization, task success probability, and response time, demonstrating good adaptability and practicality;

Compared to static scheduling and depth first, dynamic programming is more suitable for complex multi-threaded scenarios, which can further improve the reliability and real-time rendering of the system. At the same time, this article comprehensively evaluates the proposed solution from the perspectives of algorithm fault tolerance, power consumption, and installation dependencies, summarizes its specific application scope and practical application path, and has good reference significance for future practical applications. At the same time, this solution also provides an effective scheduling optimization strategy for tasks with high real-time requirements such as VR scenes and smart cities. The next step is to combine reinforcement learning with distributed architecture to enhance the practical application potential of this solution in distributed architecture scenarios such as heterogeneous intelligent terminals. In summary, the algorithm proposed in this article has demonstrated good theoretical innovation and engineering effectiveness in virtual reality multi scene scheduling optimization, and has strong academic value and application prospects.

References

[1] Xionghui L, Xiaoyu Z, Jiaming Q , et al.“Fake it, you can touch it”: a study of virtual touch effects based on VR technology[J].Virtual Reality,2024,29(1):15-15.https://dol:10.1007/s10055-024-01092-y.

[2] Lumpkin T L , Nutt C G , Folds P E ,et al.Influence of Familiarity, Aesthetic Value, and Change Type on Visual Memory of Real-World Scenes in a Virtual Reality Change/No-Change Paradigm[J].Journal of Vision, 2024, 24(10):2.https://dol: 10.1167/jov.24.10.948.

[3] Sainz M, Vicente R, Plaza J. Accelerating real-time global illumination using machine learning techniques[J]. ACM Transactions on Graphics, 2020, 39(6): 1-16. https://dol.org/10.1145/3414685.3417855.

[4] Htun N N S, Egami S ,Fukuda K .Activity scenarios simulation by discovering knowledge through activities of daily living datasets[J].SICE Journal of Control, Measurement, and System Integration,2024,17(1):87-105.https://dol:10.1080/18824889.2024.2318848.

[5] Owens J D, Houston M, Luebke D, et al. GPU computing[J]. Proceedings of the IEEE, 2008, 96(5): 879-899.https://dol.org/10.1109/JPROC.2008.917757.

[6] Li B, Tian F, Cai L, et al. Survey on deep learning for virtual reality and augmented reality[J]. IEEE Transactions on Visualization and Computer Graphics, 2022, 28(7): 2900-2919.https://dol.org/10.1109/TVCG.2021.3130997

[7] Ragan-Kelley J, Adams A, Paris S, et al. Decoupling algorithms from schedules for easy optimization of image processing pipelines[J]. ACM Transactions on Graphics, 2013, 32(4): 1-12. https://dol.org/10.1145/2461912.2461978.

[8] ChaeL R, LeeH, KimE. The Effects of Avatar Human-Likeness on Psychological Closeness in Virtual-Reality[J]. Psychology & Marketing,2024,42(4):1132-1145.https://dol:10.1002/mar.22168.

[9] Son D, Im B, Her J, et al. Street lighting environment and fear of crime: a simulated virtual reality experiment[J].Virtual Reality,2024,29(1):8-8.https://dol:10.1007/s10055-024-01080-2.

[10] Navarro J D, Serrano A, Malpica S. Minimally disruptive auditory cues: their impact on visual performance in virtual reality[J].The Visual Computer,2024,41(7):1-15.https://dol: 10.1007/s00371-024-03779-4.

[11] Figueroa J A, Guan J, Swearngin J, et al. Deep learning-based real-time dynamic global illumination for immersive virtual environments[J]. ACM Transactions on Graphics, 2021, 40(4): 1-12. https://dol.org/10.1145/3450626.3459833.

[12] Li X ,Wang D J ,Dudley J J , et al.Swarm manipulation: An efficient and accurate technique for multi-object manipulation in virtual reality[J].Computers & Graphics,2024,125104113-104113.https://dol:10.1016/j.cag.2024.104113.

[13] Liu Y, Ren Z, Huang J, et al. Real-time dynamic ambient occlusion using deep neural networks[J]. IEEE Transactions on Visualization and Computer Graphics, 2020, 26(7): 2468-2477.https://dol.org/10.1109/TVCG.2019.2923033.

[14] Wei L, Wang W, Zou C, et al. Efficient real-time rendering method for complex scenes using adaptive path sampling[J]. Computer Graphics Forum, 2019, 38(7): 201-211.https://dol.org/10.1111/cgf.13734.

[15] Karras T, Laine S, Aila T. A style-based generator architecture for generative adversarial networks[J]. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019: 4401-4410.https://dol.org/10.1109/CVPR.2019.00453.

[16] Fan Y, Gong M, Chen Q, et al. Real-time rendering of complex scenes via neural radiance caching[J]. ACM Transactions on Graphics, 2021, 40(6): 1-13. https://dol.org/10.1145/3478513.3480537.

[17] Li P ,Zhang X ,Hu X , et al.Theoretical model and practical analysis of immersive industrial design education based on virtual reality technology[J].International Journal of Technology and Design Education,2024,(prepublish):1-28.https://dol:10.1007/s10798-024-09946-x.

[18] OgawaY, AoukarR, LeibbrandtR , et al.Combining Unity with machine vision to create low latency, flexible and simple virtual realities[J].Methods in Ecology and Evolution,2024,16(1):126-144.https://dol: 10.1111/2041-210X.14449.

[19] Birkheim L S, Calogiuri G ,Hvalstad M , et al.Exploring the experiences of resident doctors in child and adolescent psychiatry with virtual reality-based simulation training: a qualitative study.[J].BMC health services research,2024,24(1):1443.https://dol: 10.1186/s12913-024-11941-w.

[20] Junjie Y, Cuiying Z ,Zhen L , et al.Real-Time Rendering Closure Method for Continuous Cutting of Multilevel TIN Geological Models[J].Geotechnical and Geological Engineering,2023,42(5):3269-3285.https://dol:10.1007/s10706-023-02729-6.

[21] Laine S, Karras T, Aila T.Efficient sparse voxel octrees – Analysis, extensions, and implementation[J].ACM Transactions on Graphics, 2010, 29(4): 101.https://dl.acm.org/doi/10.1145/1778765.1778803.

[22] Dachsbacher C, Stamminger M.Reflective shadow maps[J].ACM Transactions on Graphics, 2005, 24(3): 756-764.https://dl.acm.org/doi/10.1145/1073204.1073245.

[23] Zhou K, Gong M, Huang X, et al.Real-time KD-tree construction on graphics hardware[J].ACM Transactions on Graphics, 2008, 27(5): 126.https://dl.acm.org/doi/10.1145/1409060.14091111.

[24] Engel K, Hadwiger M, Kniss J M, et al.Real-time volume graphics[J].ACM Transactions on Graphics, 2004, 23(3): 722-741.https://dl.acm.org/doi/10.1145/1015706.10158011.

[25] Luebke D, Humphreys G.Is graphics hardware ready for the revolution?[J].IEEE Computer Graphics and Applications, 2002, 22(6): 18-21.https://ieeexplore.ieee.org/document/1046627.

[26] Sik-Lányi C, Pölöskei Z.Virtual reality in medicine[J].Informatica, 2018, 42(2): 209-214.https://www.informatica.si/index.php/informatica/article/view/2324.

[27] Kuk K, Jaworski W.Parallelization of dynamic programming algorithms for modern multiprocessor platforms[J].Informatica, 2020, 44(3): 411-417.https://www.informatica.si/index.php/informatica/article/view/2956.

[28] Maček A, Kukar M.A survey of parallelization techniques for dynamic programming[J].Informatica, 2015, 39(3): 333-342.https://www.informatica.si/index.php/informatica/article/view/560.