

IM HAE IN
Game Developer

Portfolio



임해인 IM HAEIN

1998.03.03. 서울시 강남구

Tel. 010-8991-9593

Email. devhaein0303@gmail.com

<https://github.com/haein0303>

2024.

한국 공학대학교 졸업

주) 게임공학과
부) 컴퓨터공학과

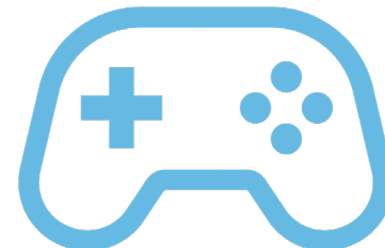
2016.

대진 디자인고등학교 졸업

컴퓨터 미디어 디자인과

Etc.

컴세바아이티탐구학원 코딩 강사 (22.11. ~)
NERU 창업동아리 대표 (2017.11 ~ 2018.11)
군) 공군 하사 전역





AlgaeEater

알지이터

장르 : 3D 액션

개발기간 : 2022.07. ~ 2023.07.

사용도구 : C++17, IOCP, DirectX12

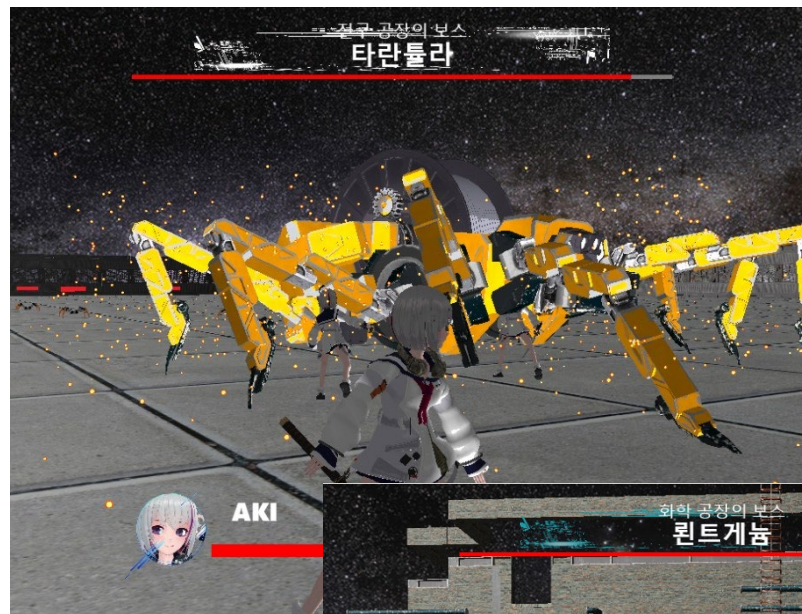
개발인원 : 3명 (클라이언트 2명, 서버 1명)

Git : <https://github.com/haein0303/AlgaeEater>

역할 :

클라이언트

- 멀티쓰레드 클라이언트
- 이동 보간
- 콘텐츠 구현



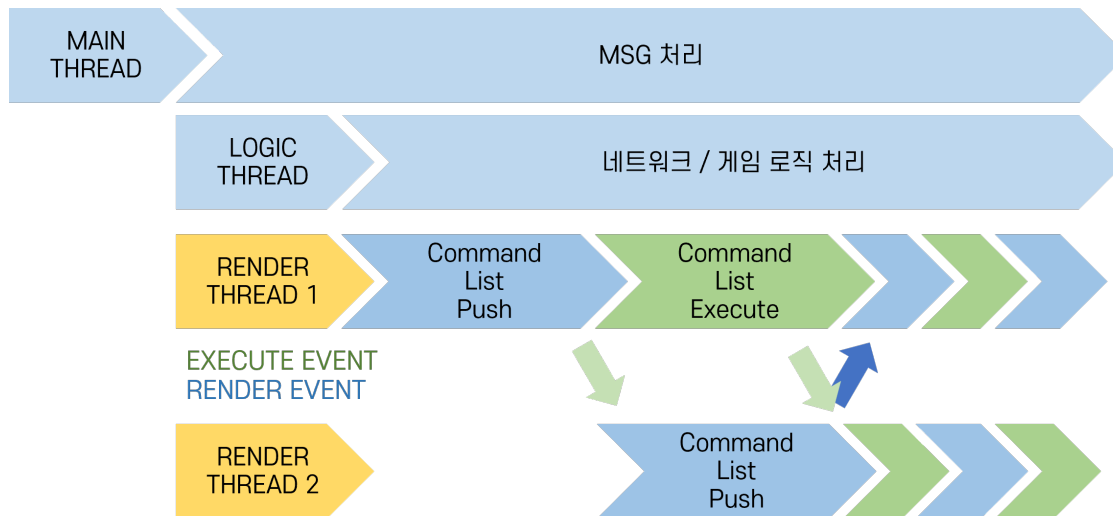
https://www.youtube.com/watch?v=LqkQf_4yFbl

DirectX12 멀티쓰레드 렌더링 구현

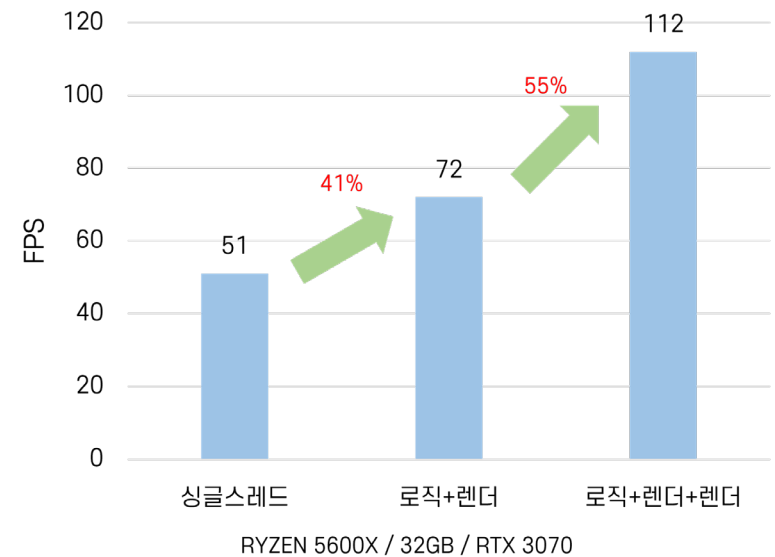
총 4개의 스레드를 사용하여, 클라이언트의 프레임을 향상

Render Thread간의 동기화는 2개의 이벤트를 사용하여 관리

결과, 싱글 스레드 대비 약 110%의 성능 향상



멀티쓰레드 렌더링 동작 소개



프레임 변화 측정

```
thread logical_thread{ &Client::Logic,&client };

thread render_thread1{ &Client::Draw,&client };
thread render_thread2{ &Client::Draw,&client };
```

```
void Draw() {

    cout << "DRAW CALL" << endl;
    int i_now_render_index;
    if (!_render_thread_num) {
        i_now_render_index = 0;
        _render_thread_num++;
    }
    else {
        i_now_render_index = 1;
        _render_thread_num = 0;
    }
    while (g_isLive) {
        ::WaitForSingleObject(dxEngine._renderEvent, INFINITE);

        dxEngine.timerPtr->TimerUpdate();

        dxEngine.timerPtr->ShowFps(windowInfo);

        dxEngine.Update(windowInfo, isActive);

        dxEngine.Draw_multi(windowInfo, i_now_render_index);
    }
}
```

로컬 변수 i_now_render_index 를 사용하여,
Atomic 변수인 i_now_render_index 에 최소한으로 접근

메소드를 스레드로 분리하여, 내부의 데이터를 사용할 수 있도록 구성

두 개의 스레드는 이벤트로 제어
각 이벤트는 command list에 push/ Execute 가 된 이후 Set

```
::WaitForSingleObject(_excuteEvent, INFINITE);
SetEvent(_renderEvent);

D3D12_RESOURCE_BARRIER barrier2 = CD3DX12_RESOURCE_BARRIER::Transition(swapChainPtr->_renderTargets[i_now_render_index].Get(), D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT);
cmdList->ResourceBarrier(1, &barrier2);
cmdList->Close();

...

swapChainPtr->_swapChain->Present(0, 0);

cmdQueuePtr->WaitSync();

swapChainPtr->_backBufferIndex = (swapChainPtr->_backBufferIndex + 1) % SWAP_CHAIN_BUFFER_COUNT;

SetEvent(_excuteEvent);
```

네트워크 최적화를 위한 이동 보간

이동 패킷의 전송량을 낮춤에 따라서, 오브젝트 이동 시 끊김 발생

끊김 방지를 위해서 이동 동기화 개발

직전 패킷부터 현재 패킷까지 프레임별로 이동 양을 제어하여 선형 보간

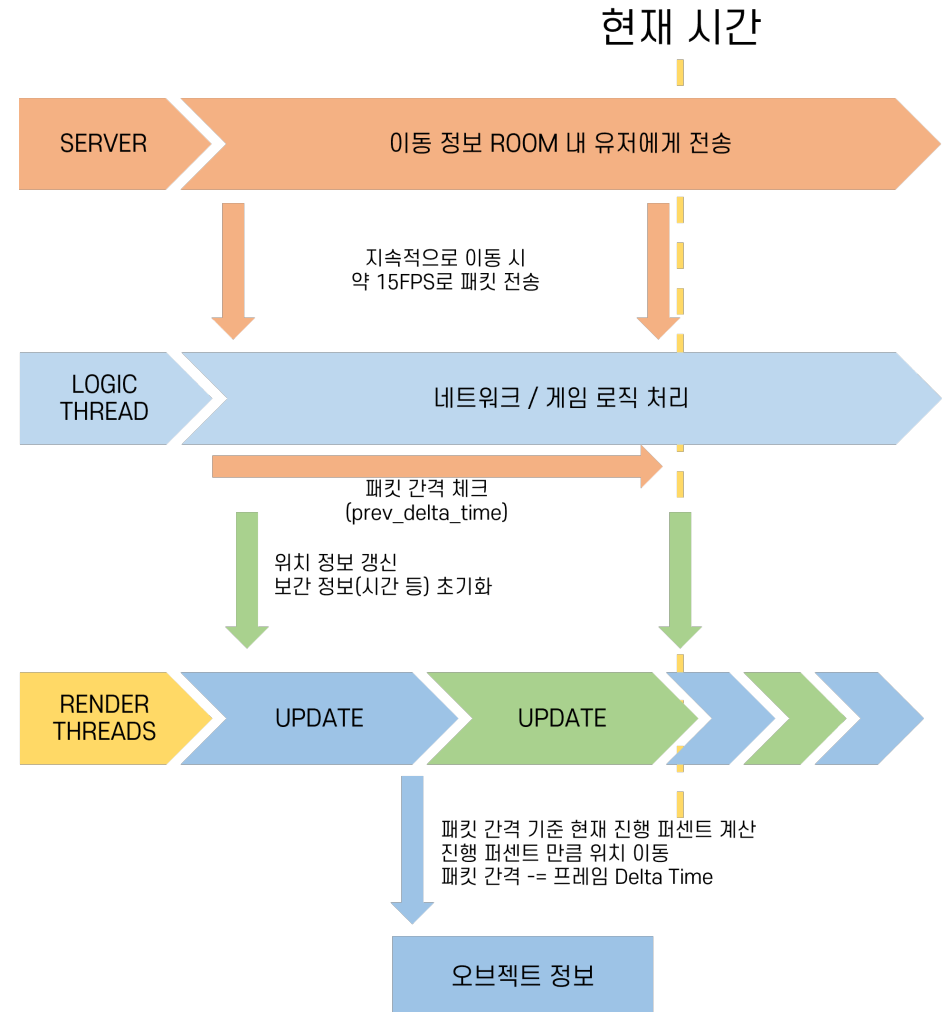
```
for (int i = 1; i < PLAYERMAX; ++i) {
    if (playerArr[i]._on) {
        float dt = timerPtr->deltaTime;
        playerArr[i]._delta_percent = dt / playerArr[i]._prev_delta_time;

        if (playerArr[i]._prev_delta_time > 0) {
            playerArr[i]._prev_delta_time -= dt;
        }
        else {
            playerArr[i]._prev_delta_time = 0;
            playerArr[i]._delta_percent = 0;
        }

        playerArr[i]._delta_transform = playerArr[i]._transform - playerArr[i]._prev_transform;
        playerArr[i]._delta_transform = playerArr[i]._delta_transform * playerArr[i]._delta_percent;

        playerArr[i]._prev_transform = playerArr[i]._prev_transform + playerArr[i]._delta_transform;
        playerArr[i]._prev_degree = playerArr[i]._degree;
    }
}
```

DxEngine.cpp / line :1322~





게임서버 텀 프로젝트

https://github.com/haein0303/GameServer_Final_term

프로젝트 소개

- IOCP를 활용한 MMORPG 게임 서버 구현

구현 내용(C++ / IOCP)

- 시야처리
- 파티
- 경험치 및 레벨
- 채팅
- NPC 로밍
- NPC Agro

파티 구현

```
if (clients[pa]._hp <= 0) { //NPC 죽음
    //나중에 재활용할 수 있으니깐
    clients[pa]._target_id = -1;

    clients[pa]._ll.lock();
    clients[pa]._state = ST_ALLOC;
    clients[pa]._ll.unlock();

    clients[c_id].send_die_packet(pa, 100);

    if (clients[c_id]._party_id > -1) {
        auto get_pair = party_map.find(clients[c_id]._party_id);
        for (int i = 0; i < MAX_PARTY; ++i) {
            if (get_pair->second._player_id[i] == -1) continue;
            printf("PARTY BONUS : %d [%d]Wn", get_pair->second._player_id[i], 20);
            clients[get_pair->second._player_id[i]].send_die_packet(c_id, 20);
        }
    }
}
```

가입과 삭제 보조 함수

배열을 재사용하기 위해서, 추가와 삭제를 담당하는 함수를 작성하였습니다.

파티 보너스

만약 오브젝트가 파티 아이디가 있다면, 파티원에게 경험치 일부를 공유합니다.


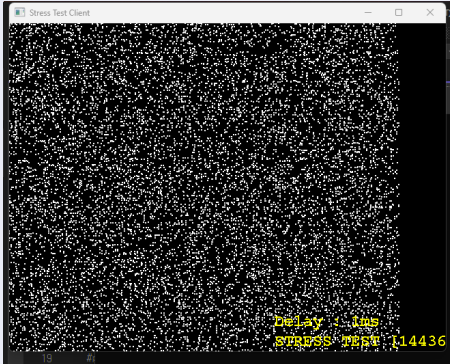
관리

파티원은 배열로 관리합니다.

컨테이너를 사용하기에 크기가 정해져 있고, 반복 횟수가 많지 않아서 배열로 처리하였습니다.

```
//조인 가능하면 1, 아니면 0
int join(int id) {
    for (int i = 0; i < MAX_PARTY; ++i) {
        if (_player_id[i] == -1) {
            _player_id[i] = id;
            return 1;
        }
    }
    return 0;
}

//성공하면 1, 아니면 0
int exit(int id) {
    for (int i = 0; i < MAX_PARTY; ++i) {
        if (_player_id[i] == id) {
            _player_id[i] = -1;
            return 1;
        }
    }
    return 0;
}
```


Multithread IOCP	최적화 이후
	
최대 동시 접속	
653	14436

최적화 수행 결과

https://github.com/haein0303/Game_server_tutorial

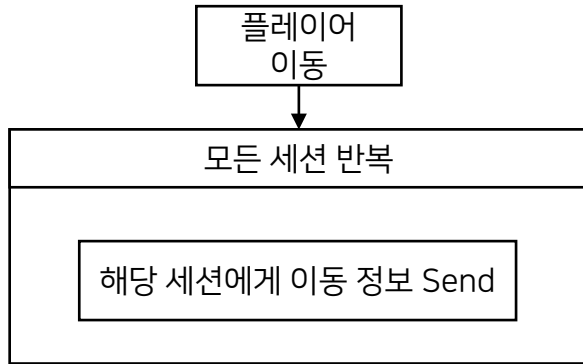
구현 의도

모든 섹션에게 데이터를 전송하는 Network overhead를 줄이기 위하여, 시야처리 및 ZONE을 구현

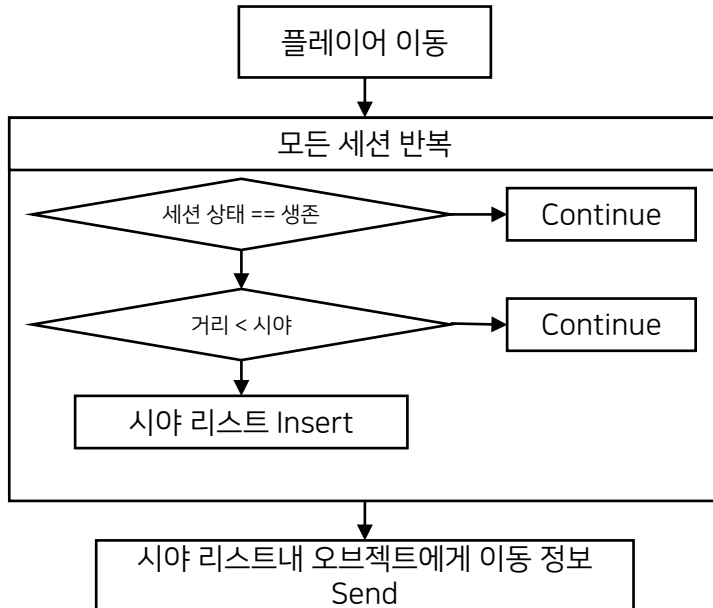
구현 내용

시야처리
ZONE 구현
Multi Thread IOCP

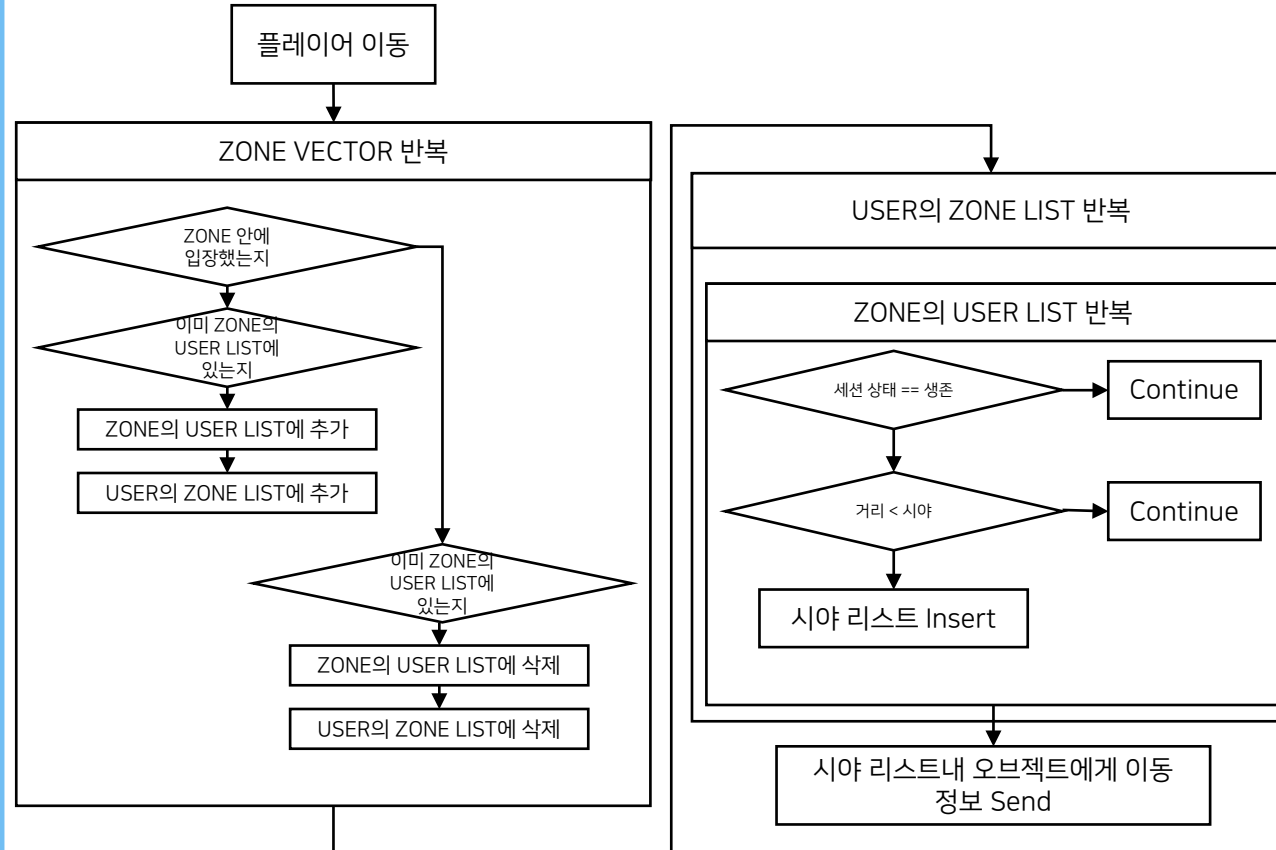
기존 MultiThread IOCP 방식



시야처리 적용



시야처리 및 ZONE 적용



ZONE 내부에 있는지 검사 및 처리 코드

```
137 bool contains(SESSION& player) {
138
139     bool is_in = false;
140     //내부에 있는지 검사
141     if (player.x >= x_min_ && player.x <= x_max_ && player.y >= y_min_ && player.y <= y_max_)
142     {
143         is_in = true;
144     };
145
146     if (is_in) {
147         //없을때 처리
148         //있을때만 검사하고 동작시켜야함
149         zl.lock();
150         if (user_list.count(player._id) == 0) {
151             //유저 zonelist에 추가
152             //내꺼도 추가
153
154             user_list.insert(player._id);
155             zl.unlock();
156             player._zl.lock();
157             player._zone_list.insert(my_num);
158             player._zl.unlock();
159             return is_in;
160         }
161         zl.unlock();
162     }
163     else {
164         //있을때 처리
165         //있을때만 검사하고 동작시켜야함
166         zl.lock();
167         if (user_list.count(player._id) != 0) {
168             //유저 zonelist에서 빼줘야함
169             //내꺼에서도 빼야함
170
171             user_list.erase(player._id);
172             zl.unlock();
173             player._zl.lock();
174             player._zone_list.erase(my_num);
175             player._zl.unlock();
176             return is_in;
177         }
178         zl.unlock();
179     }
180
181     return is_in;
182 }
183
184
185
```

검사 이후 시야리스트 갱신 및 전송 코드

```
clients[c_id]._vl.lock();
auto old_vl = clients[c_id]._view_list;
clients[c_id]._vl.unlock();

//Todo : 모든 클라이언트 검사가 아니라 zone에서 검사하자

for (auto& m : g_map.zone_list) {
    m.contains(clients[c_id]);
}

unordered_set<int> new_vl;

for (auto& vl : clients[c_id]._zone_list) {
    for (auto& p : g_map.zone_list[vl].user_list) {
        if (p == c_id) continue;
        if (can_see(c_id, p) == false) continue;
        new_vl.insert(p);
    }
}

for (auto& o : new_vl) {
    if (old_vl.count(o) == 0) {
        clients[o].send_add_player_packet(c_id);
        clients[c_id].send_add_player_packet(o);
    }
    else {
        clients[o].send_move_packet(c_id);
        clients[c_id].send_move_packet(o);
    }
}

clients[c_id].send_move_packet(c_id);
```

네트워크 overhead를 최소화 하기 위하여
시야처리를 도입하였고, 시야처리를 처리하기 위한 연산이
Overhead가 되기 때문에, ZONE을 활용하여 성능 개선

흔한 공격으로 살아남아 보자

장르 : 3D 디펜스

개발기간 : 2022. 05.

사용도구 : Unity

개발인원 : 1명

Git : https://github.com/haein0303/survive_normal_attack

게임 소개

무한으로 나오는 다양한 종류의 몬스터를 피하고,
공격하며 최대한 살아남아 기록을 갱신하는 게임입니다.

구현 내용

네비게이션 메쉬 활용 NPC 구현
트레일 렌더러 활용 공격체 구현
NPC 원거리 공격 구현

