Project report for the CG 100433 course

Project Title

Minecraft with database

Team member

- 1951443罗劲桐(GROUP LEADER)；
- 1952941鲁翔辰；
- 1853348聂熙承.

Abstract

This project is a demo of a Minecraft sandbox game. The project implements the basic block and scene generation in Minecraft, lighting rendering, collision detection logic, and placing and destroying cubes. In addition, we use the database to realize an infinite map that can be automatically generated, stored, and loaded with the help of block cache scheduling, and use perspective and surface clipping to improve rendering efficiency.

Motivation

Minecraft is a sandbox game that has been popular around the world for a few years. It is popular among game fans for its excellent gameplay and unique block style. The team members had a strong interest in the game and experienced it firsthand. Then I want to achieve a small game of demo nature of Minecraft through this group work.

In order to make this project more interesting and challenging, the game uses dynamic random generation to generate new map partitions, and uses SQL to store the map and dynamically load map blocks according to the position of the character to remove redundant maps and ensure memory loading efficiency.

We hope to realize the realization of this function by displaying the comparison between the memory load and the map cached in the database, and by creating a large batch of new map partitions.

The Goal of the project

- Realize the physics engine of Minecraft;
- Realize a simple natural light rendering；
- Realize the modeling of common blocks(plus textures) and scenes;
- Realized the placement and destruction of blocks;
- Realize the real-time generation of new map partitions;
- Realize the storage and recall of map partitions in the database with the help of block cache scheduling;
- Real-time perspective and surface clipping.

The Scope of the project

- The sky uses a fixed map, and the cloud is fixed and cannot be modified
- The project does not include the spawning and movement of other biological entities

except players.

- The project does not include animation for block placement and destruction

Involved CG techniques

- Database storage: SQLite database for map loading, storage and rendering;
- Cache scheduling: NRU cache replacement algorithm;
- Dynamic random generation: Random generation of discrete uniform distribution;
- Graphics rendering pipeline: GLFW, GLM implementation model output to GPU and rendering;
- Physics Engine: Position detection.

Project contents

The project is a demo of a Minecraft sandbox game, which will achieve the following functions:

- Random generation of maps;
- Dynamic save and recall of the map;
- Real-time perspective and surface clipping;
- Realize scenes composed of various blocks;
- Characters walking, jumping, colliding;
- Simple natural light.

Implementation

1. Block memory schedule

Use two parts to allocate all blocks. Load the block that needs to be render in the player's nearest neighbors. The blocks swapped out from the load are not immediately discarded, but stored in the cache to reduce the number of calls to the database. The data structure is:

```
1.    unsigned const int LOAD_SIZE = BLOCK_LOAD_NUM * BLOCK_LOAD_NUM * BLOCK_LOAD_NUM;
2.    unsigned const int CACHE_SIZE = (BLOCK_LOAD_NUM + 1) * (BLOCK_LOAD_NUM + 1) *
    (BLOCK_LOAD_NUM + 1) - BLOCK_LOAD_NUM * BLOCK_LOAD_NUM * BLOCK_LOAD_NUM;
3.
4.    DBLink db;
5.    std::string map_name;
6.    Block* block_load[BLOCK_LOAD_NUM * BLOCK_LOAD_NUM * BLOCK_LOAD_NUM];
7.    Block* block_cache[(BLOCK_LOAD_NUM + 1) * (BLOCK_LOAD_NUM + 1) * (BLOCK_LOAD_NUM + 1) -
    BLOCK_LOAD_NUM * BLOCK_LOAD_NUM * BLOCK_LOAD_NUM];
8.    BlockPos block_pos[(BLOCK_LOAD_NUM + 1) * (BLOCK_LOAD_NUM + 1) * (BLOCK_LOAD_NUM + 1)];
```

When actually playing, you should only display the load content as a complete block, but in order to show the load and cache exchange methods more clearly, this project renders the blocks in the load and cache at the same time and uses different colors to show them. According to the current location of the block, the cube type replacement logic:

```
1.    CUBE_TYPE cube = data[j][k][l];
2.    if (cube != CubeType::NONE)
3.              push(CubeType(int(cube) - 1), base_x + j, base_y + k, base_z + l);
```

Since the edges of the blocks are not neat, the effect of surface culling on the borders will be worse than that of neat cube rendering. However, due to the addition of surface culling and frustum culling, rendering performance is guaranteed.

2. The cache replacement algorithm

We use the NRU cache abandonment algorithm, and modifies the calling method of the adapted cache block to optimize the efficiency of cache calling.

The performance of the CLOCK algorithm is closer to that of the LRU. The NRU algorithm adds a modified bit on the basis of the used bits, and an improved CLOCK replacement algorithm is obtained, which makes the algorithm more efficient. In this way, each frame is in one of the following four situations:

1) It has not been visited recently and has not been modified (u=0, m=0)

2) It has been visited recently, but has not been modified (u=1, m=0).

3) It has not been visited recently, but has been modified (u=0, m=1)

4) Recently visited and modified (u=1, m=1)

The algorithm performs the following steps:

1) Scan the frame buffer from the current position of the pointer. During this scan, no modification is made to the used bits. The first frame encountered (u=0, m=0) is selected for replacement.

2) If step 1 fails, scan again to find (u=0, m=1) frames. The first such frame encountered is selected for replacement. In this scanning process, for each skipped frame, set its use bit to 0.

3) If step 2 fails, the pointer will return to its original position, and the used bits of all frames in the set are 0. Repeat step 1, and if necessary, repeat step 2. This will find a frame for replacement.

```
1.  size_t BlockMemFlow::nru_cache_search(const int* exist) {
2.    size_t pointer_pos = nru_pointer;
3.    int search_stage = -1;
4.    bool loop = true;
5.    nru_pointer--;
6.    while (loop)
7.    {
8.              //向前走一格
9.              nru_pointer++;
10.             if (nru_pointer >= CACHE_SIZE)
11.                       nru_pointer -= CACHE_SIZE;
12.             if (nru_pointer == pointer_pos)
13.                       search_stage++;
14.
15.             switch (search_stage)
16.             {
17.                       case 0:
18.                                 if (block_cache[nru_pointer] == nullptr ||
    (!block_cache[nru_pointer]->isDirty() && !block_cache[nru_pointer]->isUse()))
19.                                 {
20.                                           if (exist == nullptr || exist[nru_pointer] == 0)
21.                                                     loop = false;
22.                                 }
23.                                 break;
24.                       case 1:
25.                                 if (!block_cache[nru_pointer]->isUse(true) &&
    block_cache[nru_pointer]->isDirty())
```

```
                                {
                                        if (exist == nullptr || exist[nru_pointer] == 0)
                                                loop = false;
                                }
                                break;
                        case 2:
                                // 写在前防止被优化
                                if (!block_cache[nru_pointer]->isDirty()
    && !block_cache[nru_pointer]->isUse())
                                {
                                        if (exist == nullptr || exist[nru_pointer] == 0)
                                                loop = false;
                                }
                                break;
                        case 3:
                                if (block_cache[nru_pointer]->isDirty()
    && !block_cache[nru_pointer]->isUse())
                                {
                                        if (exist == nullptr || exist[nru_pointer] == 0)
                                                loop = false;
                                }
                                break;
                        default:
                                std::cout << "Error: NRU 算法没有找到可替换的块" << std::endl;
                                loop = false;
                                break;
                }
    }

    //向前走一格
    pointer_pos = nru_pointer;
    nru_pointer++;
    if (nru_pointer >= CACHE_SIZE)
                nru_pointer -= CACHE_SIZE;
    std::cout << "INFO: NRU 算法替换成功" << std::endl;
    return pointer_pos;
}
```

The advantage of the NRU algorithm over the simple CLOCK algorithm is that pages that have not changed are preferred during replacement. Since modified pages must be written back before being replaced, this saves time. At the same time, compared with the LRU algorithm, the complexity cost of NRU implementation is lower.

3. Database interaction logic

We use create, drop and other statements to create and delete maps, and use insert, select, and update statements to write, read, and update map blocks. Specifically, when calling the database, it is divided into the following two situations:

The call method with a small amount of data usually has less inserted value and returned result data, and the data is obtained and processed by the callback function:

```
    /*generate*/
    sql_cmd = std::string().append("create table '").append(map_name).append("'(block_x int,
    block_y int, block_z int, block_data blob)").append(";");
    rc = sqlite3_exec(sqlite, sql_cmd.c_str(), callback, 0, &zErrMsg);
    /*delete*/
    sql_cmd = std::string().append("drop table '").append(map_name).append("';");
    rc = sqlite3_exec(sqlite, sql_cmd.c_str(), callback, 0, &zErrMsg);
    /*find map*/
```

```
8.    sql_cmd = std::string().append("select count(*) from sqlite_master where type='table' and
    name='").append(map_name).append("';");
9.    rc = sqlite3_exec(sqlite, sql_cmd.c_str(), DBLink::callback_is_exist, 0, &zErrMsg);
10.   /*find block*/
11.   sql_cmd = std::string().append("select count(*) from '").append(map_name).append("'
    ").append(block.where_clause()).append(";");
12.   rc = sqlite3_exec(sqlite, sql_cmd.c_str(), DBLink::callback_is_exist, 0, &zErrMsg);
```

Calling methods for large amounts of data, including inserting, updating blocks and reading blocks from the database, use a two-step method, using sqlite3_stmt to bind the data block (blob) before writing and reading:

```
1.    sql_cmd = std::string().append("update '").append(map_name).append("' set block_data=?
    ").append(block.where_clause()).append(";");
2.    rc = sqlite3_prepare(sqlite, sql_cmd.c_str(), sql_cmd.length(), &stmt, nullptr);
3.    if (rc == SQLITE_OK)
4.    {
5.              rc = sqlite3_bind_blob(stmt, 1, data.read(), sizeof(CUBE_TYPE) * BLOCK_LENGTH_X
    * BLOCK_LENGTH_Y * BLOCK_LENGTH_Z, NULL);
6.              if (rc == SQLITE_OK) {
7.                        sqlite3_step(stmt);
8.                        std::cout << "Table update successfully" << std::endl;
9.              }
10.   }
```

4. Clipping

・ Frustum clipping

Obtain the view transformation matrix view and projection matrix projection before each rendering, calculate the 6 clipping planes of the viewing frustum, and enter the clipping plane equation for each square to determine whether the square is in the clipping space

Deriving the six faces of the viewing cone, each space plane equation can be expressed as:

Ax+By+Cz=0;

For the transformed v, if it is inside the frustum, there are:

-w'<x’<w',-w'<y’<w',-w'<z’<w';

For the left clipping plane -w'<x', there are:

-(v*row4)<(v*row1)

∴0<v(v*row4)+(v*row1)

∴0<v*(row4+row1)

So the equation of the left clipping plane is derived:

x(m41+m11) + y(m42+m12) + z(m43+m13) + w(m44+m14) = 0

Since w=1,

x(m41+m11) + y(m42+m12) + z(m43+m13) + (m44+m14) = 0

Obtain the plane expression, the other faces are the same.

・ Surface clipping

The surface clipping function is used to judge whether there are blocks around, and only the blocks with exposed surfaces are rendered. For each block, we render the cube which is on the boundary (j == 0 || j == BLOCK_LENGTH_X - 1 || k == 0 || k == BLOCK_LENGTH_Y - 1 || l == 0 || l == BLOCK_LENGTH_Z), or if it has an exposed surfaces (an adjacent cube ==

CubeType::NONE).

5. Rendering method

The CubeFlow class calls the drawAll function, in which the OpenGL surface culling is turned on first, as a further optimization, to avoid rendering invisible surfaces. Then call the drawStart function of the Cube static class to initialize the parameters in the shader:

```
1.  void Cube::drawStart()
2.  {
3.    shader.use();
4.    // 设置方块观察位置
5.    view = glm::lookAt(control->cameraPos, control->cameraPos + control->cameraFront,
    control->cameraUp);
6.    shader.setMat4("view", view);
7.
8.    // 方块光照
9.    glm::mat4 lightProjection, lightView;
10.   glm::mat4 lightSpaceMatrix;
11.   float near_plane = 1.0f, far_plane = 7.5f;
12.   lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane);
13.   lightView = glm::lookAt(lightPos, glm::vec3(0.0f), glm::vec3(0.0, 1.0, 0.0));
14.   lightSpaceMatrix = lightProjection * lightView;
15.
16.   shader.setMat4("lightSpaceMatrix", lightSpaceMatrix);
17.   shader.setInt("diffuseTexture", 0);
18.   shader.setInt("shadowMap", 1);
19.   shader.setVec3("viewPos", control->eyePos);
20.   shader.setVec3("lightPos", lightPos);
21. }
```

Finally, the drawAll function calls the draw function of each Cube instance, and finally completes the rendering through the OpenGL instantiation method glDrawElementsInstanced.

```
1.  void Cube::draw()
2.  {
3.    if (maxCnt == 0)
4.              return;
5.
6.    // OpenGL Instantiate
7.    glBindBuffer(GL_ARRAY_BUFFER, painterVBO_H);
8.    void* buffer = glMapBuffer(GL_ARRAY_BUFFER, GL_WRITE_ONLY);
9.
10.   memset(buffer, 0, maxCnt * sizeof(glm::mat4));
11.   memcpy(buffer, &painterVec[0], maxCnt * sizeof(glm::mat4));
12.   glUnmapBuffer(GL_ARRAY_BUFFER);
13.
14.   glBindTexture(GL_TEXTURE_2D, textureH);
15.   glBindVertexArray(VAO_H);
16.   glDrawElementsInstanced(GL_TRIANGLES, sizeof(indicesH), GL_UNSIGNED_INT, 0, maxCnt);
17.
18.
19.   // OpenGL Instantiate
20.   glBindBuffer(GL_ARRAY_BUFFER, painterVBO_V);
21.   buffer = glMapBuffer(GL_ARRAY_BUFFER, GL_WRITE_ONLY);
22.   memset(buffer, 0, maxCnt * sizeof(glm::mat4));
23.   memcpy(buffer, &painterVec[0], maxCnt * sizeof(glm::mat4));
24.   glUnmapBuffer(GL_ARRAY_BUFFER);
25.
26.   glBindTexture(GL_TEXTURE_2D, textureV);
```

```
27.     glBindVertexArray(VAO_V);
28.     glDrawElementsInstanced(GL_TRIANGLES, sizeof(indicesV), GL_UNSIGNED_INT, 0, maxCnt);
29. }
```

6. Block model and texture

Since the texture on the top of the dirt block in Minecraft is different from the surroundings (the dirt block on the ground has grass on the top), the vertex data in the Cube class is divided into two arrays of vertical vertices and horizontal vertices, and the top surface and other surfaces are respectively mapped. It can also be seen from the part of calling the Cube constructor in the CubeFlow header file that the two texture maps for the dirt square on the surface are different, and the other squares are the same:

```
1.     Cube cube1 = Cube("dirt.png", "dirt.png");
2.     Cube cube2 = Cube("sand.png", "sand.png");
3.     Cube cube3 = Cube("stone.png", "stone.png");
4.     Cube cube4 = Cube("planks_oak.png", "planks_oak.png");
5.     Cube cube5 = Cube("cobblestone.png", "cobblestone.png");
6.     Cube cube6 = Cube("wool_colored_lime.png", "dirt.png");
```

The texture map uses the loadTexture function provided by LearnOpenGL, which will not be repeated here.

7. Shader

The uniform variables in the vertex shader include the projection matrix, the view observation matrix, and the lightSpaceMatrix illumination matrix. These parameters are passed in to the Cube class, and the aPos position vector, aNormal normal vector and aTexCoords texture coordinates are passed in through the VAO. The principle is more conventional.

In terms of the lighting of the fragment shader, there is no light map for diffuse reflection and specular reflection, only the ambient light is lightmapped, and the result is obtained after calculation according to the formula. At the same time, the shadow of the square is obtained through a simple shadowMap calculation.

Results

Demonstration

Roles in group

GROUP MEMBER	IMPLEMENTATION CONTENT	CONTRIBUTION
1951443罗劲桐	Call of database, block cache scheduling, cube flow generation and clipping, map generation	100%
1952941鲁翔辰	Rendering, Block model and texture, Shader	100%
1853348聂熙承		0%

References

https://learnopengl.com/Introduction;
https://github.com/Hopson97/MineCraft-One-Week-Challenge/tree/master/Source;
http://www.linuxgraphics.cn/graphics/opengl_view_frustum_culling.html;

https://www.runoob.com/sqlite/sqlite-tutorial.html;
https://blog.csdn.net/Programmer_Dong/article/details/115678387.