Ray casting - dokumentacja

Mikołaj Kubik 291083

12 czerwca 2019

Spis treści

1 Wstęp

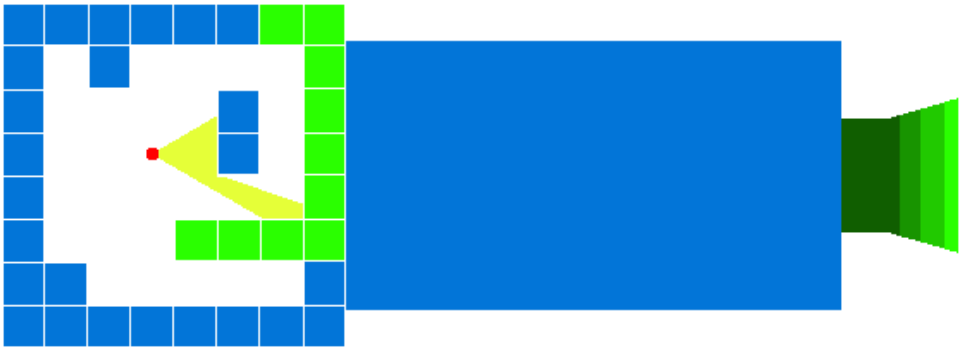
1.1 Ray casting

1.1.1 Definicja

Ray casting to technika renderowania trójwymiarowych scen na podstawie dwuwymiarowej mapy. Generowanie perspektywy 3D odbywa się za pomocą algorytmu:

- generowanie promienia wycinka koła pola widzenia kamery,
- znalezienie końca promienia (punktu przecięcia z obiektem na mapie, końca mapy lub punktu odległego od kamery o długość promienia),
- wygenerowanie fragmentu widoku na podstawie odległości między kamerą a końcem promienia.

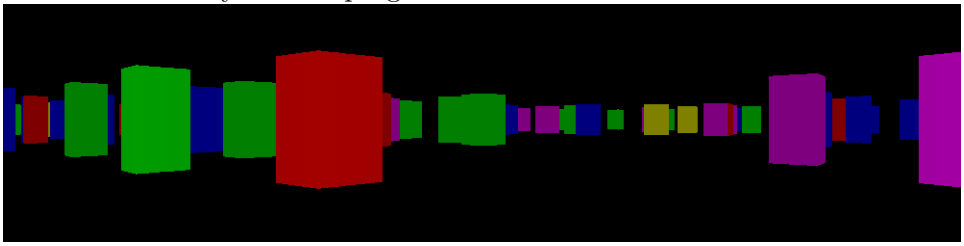
1.1.2 Schemat



1.2 Działanie programu

Zadaniem programu jest prezentacja metody raycastingu. Losowo generuje on macierz rzadko wypełnioną elementami o z góry zadanej ilości wartości, odpowiadającymi obiektom. W praktyce, z punktu widzenia użytkownika program dla każdej różnej od zera wartości macierzy generuje prostopadłościan o takiej wysokości aby stworzyć złudzenie perspektywy i w kolorze odpowiadającym tej wartości.

Zrzut ekranu z wywołania programu:



2 Użyte biblioteki

2.1 SFML

SFML(Simple and Fast Multimedia Library) to biblioteka zapewniająca interfejs do obsługi urządzeń multimedialnych. W programie wykorzystywana jest do tworzenia okna, wyświetlania obrazu oraz obsługi zdarzeń związanych z klawiaturą i myszą.

2.2 Google Test

Google Test to otwarta biblioteka pozwalająca na testowanie oprogramowania pisanego w językach C i C++.

3 Uruchomienie

Za kompilację, testowanie i czyszczenie odpowiada skrypt MakeFile, przechowujący zależności między poszczególnymi plikami.

3.1 Program

Aby utworzyć plik wykonywalny programu należy użyć polecenia

```
~ RayCasting git:(master) make
```

W odpowiedzi powinny pojawić się polecenia wykonywane aby ostatecznie utworzyć plik wykonywalny *raycasting*.

```
g++ main.cpp -c -o main.o
g++ Map.cpp -c -o Map.o
g++ View.cpp -c -o View.o
g++ Actor.cpp -c -o Actor.o
g++ Raycaster.cpp -c -o Raycaster.o
g++ main.o Map.o View.o Actor.o Raycaster.o -lsfml-
    graphics -lsfml-window -lsfml-system -g -o
    raycasting
```

Program może teraz zostać uruchomiony. Aby wyłączyć program należy użyć klawisza **ESC**.

3.2 Testy

Aby uruchomić testowanie należy użyć polecenia make z parametrem test. Odpowiednie pliki zostaną skompilowane oraz uruchomiony zostanie nowo utworzony plik wykonywalny *test*.

```
~ RayCasting git:(master) make test

g++ tests.cpp -c -o tests.o
g++ tests.o Map.o View.o Actor.o Raycaster.o
-lsfml-graphics -lsfml-window
-lsfml-system -pthread -lgtest -g -o test
./test

...

[----------] Global test environment tear-down
[==========] 9 tests from 3 test cases ran.
[  PASSED  ] 9 tests.
```

Po wykonaniu polecenia w oknie konsoli powinny wyświetlić się wyniki testów jednostkowych.

3.3 Czyszczenie

Pliki wygenerowane podczas kompilacji programu możemy usunąć za pomocą polecenia make z parametrem clean. Uruchomi ono skrypt z pliku makefile usuwający wszystie pliki z rozszerzeniem .o oraz pliki wykonywalne raycasting i test.

```
~ RayCasting git:(master) make clean

rm -f *.o raycasting test
```

4 Implementacja

Program składa się z czterech podstawowych klas oraz dwóch plików zawierających funkcję *main()*.

4.1 Omówienie poszczególnych klas

4.1.1 Map

Klasa zawierająca informacje o przestrzeni, która ma zostać wygenerowana w postaci widoku 3D i po której może poruszać się kamera.

Pola :

```
int m_width;
int m_height;
int **m_map;
```

Klasa Map zawiera macierz, która zostaje wypełniona zerami oraz wymiary tej macierzy.

Metody : Głównymi metodami klasy Map są metody dostępowe, pilnujące aby odwoływać się jedynie do prawidłowych elementów macierzy:

```
int Map::get(int x, int y)
{
    if (x >= 0 && x < m_width && y >= 0 && y <
       m_height)
    {
        return m_map[x][y];
    }

    return -1;
}

void Map::set(int x, int y, int value)
{
    if (x >= 0 && x < m_width && y >= 0 && y <
       m_height)
    {
        m_map[x][y] = value;
    }
}
```

Klasa posiada także metodę

```
void Map::processEvents(std::list<point_t> *points)
{
    point_t point;
    while (points->size() > 0)
    {
        point = points->front();
        points->pop_front();
    }
}
```

która otrzymuje listę punktów, w których miały miejsce zdarzenia myszy aby umożliwić implementację interakcji z otoczeniem.

4.1.2 Actor

Klasa zawierająca informacje o aktorze/kamerze, która porusza się po wygenerowanym widoku.

Enum :

```
enum direction
{
    LEFT,
    RIGHT,
    UP,
    DOWN
};
```

jest wykorzystywany w celu przekazania informacji w którym kierunku ma przemieścić się aktor.

Pola :

```
public:

double m_x{0}; // pozycja na mapie, wzpolzedna X
double m_y{0}; // pozycja na mapie wspolzedna Y

private:

int m_view_angle{90}; // pole widzenia
double m_heading{0}; // kierunek w ktory zwrocona jest
    kamera
double m_rotation_speed{1}; // predkosc obrotu kamery
double m_movement_speed{1}; // predkosc
   przemieszczania sie
Map *m_map; // mapa, po ktorej aktor sie porusza
```

Współrzędne kamery opisywane są za pomocą zmiennych typu double aby zapewnić płynność ruchu przy niskiej rozdzielczości mapy.

Metody : Metoda move:

```
void Actor::move(direction direction)
{
    double x = m_x;
    double y = m_y;
    switch (direction)
    {
    case UP:
            x = m_x + m_movement_speed * cos(m_heading
                * M_PI / 180);
            y = m_y + m_movement_speed * sin(m_heading
                * M_PI / 180);
```

```
            break;

        case DOWN:
                x = m_x + m_movement_speed * cos((
                    m_heading + 180) * M_PI / 180);
                y = m_y + m_movement_speed * sin((
                    m_heading + 180) * M_PI / 180);
            break;

        case LEFT:
                x = m_x + m_movement_speed * cos((
                    m_heading - 90) * M_PI / 180);
                y = m_y + m_movement_speed * sin((
                    m_heading - 90) * M_PI / 180);
            break;

        case RIGHT:
                x = m_x + m_movement_speed * cos((
                    m_heading + 90) * M_PI / 180);
                y = m_y + m_movement_speed * sin((
                    m_heading + 90) * M_PI / 180);
            break;

        default:
            break;
        }

        if (x > 0 && x < m_map->getWidth() && y > 0 && y <
            m_map->getHeight() && m_map->get((int)x, (int)
           y) == 0)
        {
            m_x = x;
            m_y = y;
        }
}
```

odpowiada za zmianę położenia aktora w zależności od kierunku, w który jest zwrócony oraz za zatrzymanie ruchu w momencie, gdy napotkana zostanie przeszkoda.
Metoda rotate:

```
void Actor::rotate(direction dir)
{
    switch (dir)
    {
```

```
        case LEFT:
            m_heading - m_rotation_speed >= 0 ? m_heading
               -= m_rotation_speed : m_heading = 360 -
                m_rotation_speed;
            break;

        case RIGHT:
            m_heading + m_rotation_speed <= 360 ?
                m_heading += m_rotation_speed : m_heading =
                 0 + m_rotation_speed;
            break;
    }
}
```

odpowiada za obrót kamery w odpowiednim kierunku. Kierunek kamery wyrażony jest w stopniach w przedziale od 0 do 360.
Metoda processEvents:

```
void Actor::processEvents(std::list<int> *keys)
{
    while (keys->size() > 0)
    {
        switch (keys->front())
        {
        case 0:
            rotate(LEFT);
            break;

        case 3:
            rotate(RIGHT);
            break;

        case 71:
            move(LEFT);
            break;
        case 72:
            move(RIGHT);
            break;

        case 73:
            move(UP);
            break;

        case 74:
            move(DOWN);
```

```
                break;

            default:
                break;
            }
            keys->pop_front();
        }
}
```

jako argument przyjmuje listę wartości oddpowiadających naciśniętym przyciskom. W tej metodzie zdefiniowane są zależności pomiędzy wciśniętym przyciskiem a akcją, jaka ma zostać wykonana.

4.1.3 View

Klasa View jest wyodrębnioną klasą odpowiedzialną za wyświetlanie obrazu oraz rejestrowanie interakcji z użytkownikiem.

Pola :

```
int m_window_width{800};
int m_window_height{600};
std::vector<sf::Color> m_colors;
```

Klasa zawiera informacje o wielkości okna, jakie ma wyświetlać oraz o kolorach, które mogą posiadać wyświetlane obiekty.

Metody : Klasa view jest zaimplementowana z wykorzystaniem biblioteki SFML. Część z metod bezpośrednio odpowiada metodom sfml jednak zostały zamknięte w dodatkowych aby ułatwić zmianę narzędzia do generacji obrazu.
Metoda checkEvents:

```
void View::checkEvents(std::list<int> *keys, std::list
    <point_t> *points)
{
    sf::Event event;
    while (window.pollEvent(event))
    {
        switch (event.type)
        {
        case sf::Event::Closed:
            window.close();
            break;

        case sf::Event::KeyPressed:
```

```
            if (event.key.code == sf::Keyboard::Escape
                )
            {
                window.close();
            }
            break;

        case sf::Event::MouseButtonPressed:
            point_t clicked;
            clicked.x = event.mouseButton.x;
            clicked.y = event.mouseButton.y;
            points->push_back(clicked);
            break;

        default:
            break;
        }
}

if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up)){
    keys->push_back(sf::Keyboard::Up);
}

if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
   {
    keys->push_back(sf::Keyboard::Down);
}

if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
   {
    keys->push_back(sf::Keyboard::Left);
}

if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right)
   ){
    keys->push_back(sf::Keyboard::Right);
}

if(sf::Keyboard::isKeyPressed(sf::Keyboard::A)){
    keys->push_back(sf::Keyboard::A);
}

if(sf::Keyboard::isKeyPressed(sf::Keyboard::D)){
    keys->push_back(sf::Keyboard::D);
```

```
    }

}
```

odpowiada za wstępne przetworzenie (wyjście z aplikacji) i eksportowanie zdarzeń związanych z klawiaturą i myszą. Biblioteka sfml gromadzi informacje o zdarzeniach a metoda zajmuje się ich interpretacją na potrzeby programu. Zdarzenia związane z przemieszczaniem aktora są eksportowane na podstawie informacji czy klawisz jest wciśnięty natomiast pozostałe zdarzenia klawiaturowe reagują jedynie na naciśnięcie. Rozwiązanie to pozwala na płynniejsze i bardziej intuicyjne sterowanie.
Metoda paint line:

```
void View::paint_line(double x1, double y1, double x2,
    double y2, int color_index)
{
    double dist = y2 - y1;
    double mul = 1;

    if(dist < 10){
        mul = 0.5;
    }
    else{
        mul = dist/100 + 0.5;
    }

    mul > 1 ? mul = 1 : true;

    sf::Color color;

    if(color_index == -1){
        color = sf::Color(255, 255, 255, 255);
    }
    else if(color_index > m_colors.size()){
        color = sf::Color(0, 0, 0, 0);
    }
    else{
        color = m_colors[color_index - 1];
    }

    sf::VertexArray line(sf::Lines, 2);
    line[0].position = sf::Vector2f(x1, y1);
    line[0].color = sf::Color(color.r*mul, color.g*mul
        , color.b*mul, 255);
    line[1].position = sf::Vector2f(x2, y2);
```

```
    line[1].color = sf::Color(color.r*mul, color.g*mul
        , color.b*mul, 255);

    window.draw(line);
}
```

służy do rysowania odcinka między zadanymi punktami i o zadanym kolorze. Dodatkowo, ponieważ algorytm ray castingu wymaga rysowania jedynie pionowych odcinków, odległość pomiędzy współrzędnymi y obu punktów wykorzystywana jest do określenia jak blisko kamery znajduje się dany odcinek. Im dalej od kamery tym ciemniejszy jest generowany odcinek, aby zasymulować problemy z dostrzeganiem szczegółów oddalonych obiektów. Metoda paint pixel:

```
void View::paint_pixel(double x, double y, int
    color_index){
    sf::Color color;
    if(color_index > m_colors.size()){
        color = sf::Color(0, 0, 0, 0);
    }
    else{
        color = m_colors[color_index - 1];
    }

    sf::VertexArray point(sf::Points, 1);
    point[0].position = sf::Vector2f(x, y);
    point[0].color = color;

    window.draw(point);
}
```

pozwala na pokolorowanie pojedynczego pixela na ekranie, dzięki czemu może zostać wykorzystana do rozbudowy interfejsu użytkownika.

4.1.4 Raycaster

Pola :

Metody :

4.2 Omówienie głównego pliku programu - main.cpp

5 Testy