1. Tytuł projektu i autorzy projektu

Tytuł : Wizualizacja pola wektorowego

Autorzy:
- Jan Bizoń
- Maksym Kazhaiev
- Michał Rogowski

2. Opis projektu

Celem projektu jest napisanie programu, który pozwala wizualizować funkcję , gdzie V oraz r to wektory w przestrzeni 3D.

3. Założenia wstępne przyjęte w realizacji projektu

Program nie musi umieć interpretować żadnych funkcji wpisywanych przez użytkownika. Zestaw kilku przykładowych funkcji powinien zostać wpisany na stałe do kodu programu, a użytkownik powinien mieć możliwość wyboru jednej z nich i ewentualnie zmiany jej parametrów. Użytkownik ma możliwość zapisywania wygenerowanego obrazu do pliku PNG. Poza tym w programie zaimplementowano dwa sposoby wyświetlania strzałek oraz płaszczyznę usprawniającą prace programu w momencie gdy wyświetlana jest ich duża ilość.

4. Analiza projektu

- specyfikacja danych wejściowych

Strzałka którą rysujemy przechowujemy w pliku arrow1.geo. Plik geo przechowuje dane w postaci x1, y1, z1 - x2, y2,z2 - RGB. Każde równanie ma swój obraz , ich nazwy to : eq1.png, eq2.png, eq3.png. Wymagane jest to aby obrazy te były przechowywane w formacie png.

- opis oczekiwanych danych wyjściowych

Danymi wyjściowymi jest obraz pola wektorowego zapisany w postaci png.

- zdefiniowanie struktur danych

Kształt strzałek po wczytaniu przechowywane są w std::vector znajdującym jako pole w oknie głównym.
W tym samy miejscu przechowywane są dostępne funkcje jako std::vector<Function>.
Główne okno przechowuje również wskaźniki do okien dialogowych.

- specyfikacja interfejsu użytkownika

Użytkownik może przekazywać parametry wyświetlania za pomocą okna dialogowego pod przyciskiem F1. Może on również modyfikować parametry wyświetlanych funkcji w oknie dialogowym pod przyciskiem F2. F3 pozwala zapisywać zrzut ekranu. F4 i F5 przechowują informacje o programie.

- wyodrębnienie i zdefiniowanie zadań

- Funkcje pola wektorowego wybierane z listy (możliwość manipulacji parametrami funkcji).
- Wizualizacja pola poprzez rysowanie strzałek
- Możliwość określenia przedziału zmiennności oraz ilości odcinków na ile dzielimy ten przedział
- Możliwość zmianny długości strzałek za pomocą suwaka
- (Rozszerzenie) Automatyczny dobór długości strzałek

- (Rozszerzenie) Alternatywny sposób wyświetlania strzałek - stała długość + kolor
- (Rozszerzenie) Możliwość zapisu wyniku na dysku w postaci bit mapy
- (Rozszerzenie) Odcięcie części pola za pomocą płaszczyzny

- decyzja o wyborze narzędzi programistycznych

Do realizacji projektu wykorzystano bibliotekę wxwidgets oraz program VS z kompilatorem LLVM. Powodem tego jest dobra znajomość i wcześniejsze ich wykorzystanie.

5. Podział pracy i analiza czasowa

1. Jan Bizoń - 2 pierwsze punkty
2. Maksym Kazhaiev - 3 kolejne
3. Michał Rogowski - pozostałe

6. Opracowanie i opis niezbędnych algorytmów

```
// uzupelnienie pamieci strzalkami wedlug parametrow
void create_space(Config& config, std::vector<Segment>& data, std::vector<Segment>& arrow) {
    double min_size = 2000;
    double max_size = 0;
    for(double z = config.GetZ_Min(); z < config.GetZ_Max(); z += config.GetCutLen())
        for(double x = config.GetX_Min(); x < config.GetX_Max(); x += config.GetCutLen())
            for (double y = config.GetY_Min(); y < config.GetY_Max(); y += config.GetCutLen()) {
                double dir_x, dir_y, dir_z;
                dir_x = x, dir_y = y, dir_z = z;

                if (config.GetCurrentFun()) {
                    config.GetCurrentFun()->Transform(dir_x, dir_y, dir_z); // Wyliczenie długosci strzałki w danym kierunku (x,y,z)->(v_x, v_y, v_z)
                    double r = sqrt(dir_x * dir_x + dir_y * dir_y + dir_z * dir_z);
                    if (r == NAN || r > 5.) r = 0.0; // pozbycie sie artefaktow (przy m.in. dzieleniu przez zero)
                    if (r < min_size) min_size = r; // wartości niezbędne w momencie wyliczania koloru
                    if (r > max_size) max_size = r;

                    add_arrow(config, Point(x, y, z), Point(dir_x, dir_y, dir_z), arrow, data); // transformacja strzałki i dodanie jej do przestrzeni
                }
            }
    // ustalanie koloru - (....color.R = r_danej_strzalki  w tym momencie)
    if (config.GetPrintOption() == 0) {
        for (auto& point : data) {
            point.color.R = (point.color.R - min_size) <= ((max_size - min_size)/2.) ?
                 255. * 2. * ((point.color.R - min_size) / (max_size - min_size)) : 255;
            point.color.B = (point.color.B - min_size) > ((max_size - min_size) / 2.) ?
                255. * 2 * (1. - (point.color.B - min_size) / (max_size - min_size)) : 255;
        }
    }
}
```

```cpp
// Odpowiednie przetransformowanie strzalki i dodanie do pamieci
void add_arrow(Config& config, Point& position, Point& direction, std::vector<Segment>& arrow, std::vector<Segment>& data) {
    double eps = 1e-5;
    double min_size = 2000;
    double max_size = 0;

    double r = sqrt(direction.x * direction.x + direction.y * direction.y + direction.z * direction.z);
    double p1 = direction.x > 0 ? asin(direction.y / r) : -asin(direction.y / r) + 3.14;    // kąt 1
    double p2 = direction.x > 0 ? -asin(direction.z / r) : asin(direction.z / r);           // kąt 2
    double cut_len = config.GetCutLen();                                                    // odległość między dwiema przedziałkami
    double max_len = config.GetCurrentFun()->GetMax();                                      // długość najdłuższego wektora
    double scale = cut_len * r / max_len;                                                   // skala strzałki
    p2 = fabs(direction.x) < eps ? atan(direction.z/ direction.x) : p2;                     // korekta kąta w zerze
    p2 = fabs(direction.z) < eps ? -asin(direction.z / r) : p2;

    for (int i = 0; i < arrow.size(); i++) {
        Segment point = arrow[i];

        Multiplication_Matrix leng = Multiplication_Matrix(config.GetArrowsLen() / 100.0, config.GetArrowsLen() / 100.0, config.GetArrowsLen() / 100.0);
        ZRotate_Matrix z_rotate = p1;
        YRotate_Matrix y_rotate = p2;
        transform_line(leng, &point); // mnozenie segmentu przez macierz transformacji

        if (config.GetPrintOption() == 1) {
            Multiplication_Matrix arr_leng = Multiplication_Matrix(scale, scale, scale);
            transform_line(arr_leng, &point);
        }
        else if (config.GetPrintOption() == 0) {
            Color color(r, 0, r);           // ustawienie koloru jako dlugosc wektora - (przygotowanie do dalszego wyliczenia).
            point.color = color;
        }

        // odpowiednie ustawienie strzalki - kierunek i polozenie punktu startowego
        transform_line(z_rotate, &point);
        transform_line(y_rotate, &point);
        point += position;
        data.push_back(point); // dodanie do tablicy przechowujacej wszystkie strzalki
    }
}
```

```cpp
void draw_space(Config& config, std::vector<Segment>& data, wxPanel* draw_canvas, wxBitmap& _pic, wxImage& MyImage) {

    wxClientDC _dc(draw_canvas);
    wxBufferedDC dc(&_dc, _pic);

    dc.SetBackground(*wxGREY_BRUSH);
    dc.Clear();

    Matrix4 m7;
    m7.data[0][0] = draw_canvas->GetSize().GetWidth() / 2;
    m7.data[1][1] = -draw_canvas->GetSize().GetHeight() / 2;
    m7.data[0][3] = draw_canvas->GetSize().GetWidth() / 2;
    m7.data[1][3] = draw_canvas->GetSize().GetHeight() / 2;

    Matrix4 transform2 = m7 * m6; // transformacja obiektu na oknie - prz. swiat - okno

    // wyliczenie punktu najblizszego i najdalszego - (do celu rysowania na pasku odciecia)
    std::array<Point, 8> box = {
        Point(config.GetX_Min(), config.GetY_Min(), config.GetZ_Min()),
        Point(config.GetX_Min(), config.GetY_Max(), config.GetZ_Min()),
        Point(config.GetX_Min(), config.GetY_Min(), config.GetZ_Max()),
        Point(config.GetX_Min(), config.GetY_Max(), config.GetZ_Max()),
        Point(config.GetX_Max(), config.GetY_Min(), config.GetZ_Min()),
        Point(config.GetX_Max(), config.GetY_Max(), config.GetZ_Min()),
        Point(config.GetX_Max(), config.GetY_Min(), config.GetZ_Max()),
        Point(config.GetX_Max(), config.GetY_Max(), config.GetZ_Max()),
    };

    double max = -1000;
    double min = 1000;

    for (Point& point : box) {
        Vector4 vec, result;
        vec.Set(point.x, point.y, point.z);
        result = transform1 * vec;

        double dist = result.GetZ();
        if (dist < min) {
            min = dist;
        }
        else if (dist > max) {
            max = dist;
        }
    }

    //skala wprowadzona aby wszystko prawidlowo sie rysowalo
    double bar_scale = 5;
    config.SetFurthest(bar_scale * max);
    config.SetNearest(bar_scale * min);
```

```
// rysowanie strzalek z pamieci
for (Segment segment : data) {
    transform_line(transform1, &segment);
    Vector4 startPoint, endPoint;
    startPoint.Set(segment.begin.x, segment.begin.y, segment.begin.z);
    endPoint.Set(segment.end.x, segment.end.y, segment.end.z);

    // Odciecie
    if (config.isPlaneEnable() && startPoint.GetZ() > config.GetFarPlane()) {
        continue;
    }

    double r = segment.color.R;
    double g = segment.color.G;
    double b = segment.color.B;
    dc.SetPen(wxPen(wxColour(r, g, b)));

    // usuniecie strzalek spoza obszaru widocznosci i dostowanie jej dlugosci
    double clipping = -2.0;
    if ((startPoint.GetZ() > clipping && endPoint.GetZ() <= clipping) || (endPoint.GetZ() > clipping && startPoint.GetZ() <= clipping)) {

        Vector4 temp1 = endPoint.GetZ() <= clipping ? endPoint : startPoint;
        Vector4 temp2 = endPoint.GetZ() <= clipping ? startPoint : endPoint;
        double r = abs((clipping - temp1.data[2]) / (temp2.data[2] - temp1.data[2]));
        temp1.data[0] = (temp2.data[0] - temp1.data[0]) * r + temp1.data[0];
        temp1.data[1] = (temp2.data[1] - temp1.data[1]) * r + temp1.data[1];
        temp1.data[2] = clipping;

        startPoint = transform2 * temp1;
        endPoint = transform2 * temp2;

        startPoint.data[0] /= startPoint.data[3];
        startPoint.data[1] /= startPoint.data[3];
        startPoint.data[2] /= startPoint.data[3];

        endPoint.data[0] /= endPoint.data[3];
        endPoint.data[1] /= endPoint.data[3];
        endPoint.data[2] /= endPoint.data[3];

    }
```

```
        else if (startPoint.GetZ() <= clipping && endPoint.GetZ() <= clipping) {
            continue;
        }
        else {
            startPoint = transform2 * startPoint;
            endPoint = transform2 * endPoint;

            startPoint.data[0] /= startPoint.data[3];
            startPoint.data[1] /= startPoint.data[3];
            startPoint.data[2] /= startPoint.data[3];

            endPoint.data[0] /= endPoint.data[3];
            endPoint.data[1] /= endPoint.data[3];
            endPoint.data[2] /= endPoint.data[3];
        }
        // funkcja rysujaca
        dc.DrawLine(startPoint.GetX(), startPoint.GetY(), endPoint.GetX(), endPoint.GetY());
    }
}
```

7. Kodowanie

Zdefiniowano w osobnym pliku.

8. Testowanie

- testy niezależnych bloków
- testy powiązań bloków
- testy całościowe

9. Wdrożenie, raport i wnioski

Wyświetlanie na osiach (tzn. 0X , 0Y, 0Z) nie działa do końca poprawnie. Można na pewno popracować nad wydajnością , nie działą to najlepiej. Poza tym funkcjonalnościami które moża dodać to: wybór kształtu strzałek , wybór kolorów wyświetlania.