МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Підсумкова робота**

з дисципліни « Оптимізація та просування сайтів (SEO) »

*назва дисципліни*

на тему: « Візуалізація прямої задачі кінематики »

Виконав: студент 3 курсу групи № 632п

освітньої програми

121 інженерія програмного забезпечення

(шифр і назва ОП)

Петренко Є.О.

(прізвище й ініціали студента)

Прийняв: доц. каф. 603

Лучшев П.О.

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2024

# ЗМІСТ

[1 ПОСТАНОВКА ЗАДАЧІ 4](#_Toc186043207)

[1.1 Системна інформація 4](#_Toc186043208)

[2 МАТЕМАТИЧНА МОДЕЛЬ КІНЕМАТИЧНОЇ СХЕМИ 5](#_Toc186043209)

[2.1 Афінні перетворення 5](#_Toc186043210)

[2.2 Декомпозиція на сегменти 5](#_Toc186043211)

[2.2.1 Математична модель сегмента «c» 6](#_Toc186043212)

[2.2.2 Математична модель сегмента «a» 6](#_Toc186043213)

[2.2.3 Математична модель сегмента «b» 6](#_Toc186043214)

[2.3 Фізичні обмеження моделі 7](#_Toc186043215)

[2.4 Загальна тривимірна математична модель 7](#_Toc186043216)

[3 ПРОГРАМНА РЕАЛІЗАЦІЯ КІНЕМАТИЧНОЇ СХЕМИ 9](#_Toc186043217)

[3.1 Початкові дані 9](#_Toc186043218)

[3.2 Прив'язка системи координат до видимої області вікна 10](#_Toc186043219)

[3.3 Керування об'єктом моделювання та організація інтерфейсу 10](#_Toc186043220)

[3.4 Осі координат 12](#_Toc186043221)

[3.5 Виведення сегмента 12](#_Toc186043222)

[3.6 Використання квадратичних об’єктів 13](#_Toc186043223)

[3.7 Порядок виконання афінних перетворень 14](#_Toc186043224)

[4 РЕЗУЛЬТАТИ ВІЗУАЛІЗАЦІЇ КІНЕМАТИЧНОЇ СХЕМИ 15](#_Toc186043225)

[5 РЕАЛІЗАЦІЯ ТА КОНТРОЛЬ ВИМОГ ДО РОБОТИ 17](#_Toc186043226)

[ВИСНОВОК 18](#_Toc186043227)

[ДОДАТОК Лістинг програми 19](#_Toc186043228)

# ПОСТАНОВКА ЗАДАЧІ

Розробити застосунок для виведення на екран моделі маніпулятора, заданої кінематичною схемою і відносними розмірами (a = 0.6, b = 1.2, c = 0.54). Для керування маніпулятором та/або зміни точки спостереження необхідно використовувати обробники подій клавіатури та/або маніпулятора «миша», у яких змінюють величину відповідних параметрів:

* кут θ для обертання навколо осі OZ;
* відстань S для зміни кутів ψ, θ;
* кут aw для обертання навколо осі OY.

|  |  |  |
| --- | --- | --- |
| 16 | a = 0.8  b = 0.6  c = 0.54 |  |

## Системна інформація

Для розробки та виконання практичних робіт використовувалися наступні апаратні та програмні засоби:

Processor AMD Ryzen 5 4600H @ 3.0Ghz

RAM 16.0 GB (15.4 GB usable)

System type 64-bit operating system, x64-based processor

Edition Windows 10 Pro Version 22H2

IDE Microsoft Visual Studio Enterprise 2022 (64-bit) version 17.11.3

# МАТЕМАТИЧНА МОДЕЛЬ КІНЕМАТИЧНОЇ СХЕМИ

Розв'язання цієї задачі може бути виконане поетапно. На першому етапі задача розглядається як двомірна тільки в одній площині **Z = const = 0**, а вже на другому етапі розв'язання задачі розглядається у просторі.

## Афінні перетворення

Для реалізації поставленої задачі на площині достатньо використовувати комбінацію з двох афінних перетворень: обертання та перенесення. Для заданого варіанту необхідно використовувати обертання навколо осі Z:



та перенесення вздовж осі Y:



Для реалізації завдання у просторі додатково знадобиться обертання навколо осі Y:



обертання навколо осі X:



та масштабування по всіх трьох осях:



## 

## Декомпозиція на сегменти

Фактично всі кінематична схема може бути представлена у вигляді трьох сегментів, які для спрощення можна подати у вигляді відрізків Segment заданого розміру size, кожен з яких у початковому стані займає вертикальне положення (вздовж осі OY):



Таким чином, для вирішення завдання потрібно визначити набір афінних перетворень, які необхідні для перетворення з початкового стану на положення, що задане кінематичною схемою.

## Математична модель сегмента «c»

Розташування сегмента c всіх сегментів і для відображення у складі кінематичної схеми позначимо його як Sc, що залежить від повороту відносно осі Z на кут theta:

|  |  |
| --- | --- |
| Sc(theta)= Rz(theta)×Segment(c) | (2.7) |

## Математична модель сегмента «a»

Положення сегмента a залежить від двох кутів: theta та phi. При цьому кут phi, у свою чергу, залежить від величини S. Тому для обчислення кута phi розглянемо трикутник, який утворено сторонами a, S і c, в якому один з внутрішніх кутів є суміжним з кутом phi. Тоді можна скористатися теоремою косинусів, задаючи кут суміжний з phi зв'язавши його наступним виразом зі сторонами a, S і c:

|  |  |
| --- | --- |
| S2 = a2 + c2 - 2 \* a \* c \* cos(phi) | (2.8) |

з якого можна вивести залежність кута phi від значення S:

|  |  |
| --- | --- |
|  | (2.9) |

Таким чином, для представлення сегмента у складі кінематичної схеми введемо для сегмента позначення Sa, положення якого залежить від повороту відносно осі Z на кут theta і величини S:

|  |  |
| --- | --- |
| Sa(theta, S)= Rz(theta)×Rz(-phi(S))×Segment(a) | (2.10) |

## Математична модель сегмента «b»

Положення сегмента b залежить від трьох кутів: theta, phi та psi. При цьому кут psi, у свою чергу, залежить від величини S. Тому, для обчислення кута розглянемо трикутник, який утворено сторонами a, S і c і містить кут psi. Використовуючи теорему косинусів, наступним виразом можна зв'язати кут psi зі сторонами a, S і c:

|  |  |
| --- | --- |
| c2 = a2 + S2 - 2 \* a \* S \* cos(psi) | (2.11) |

з якого можна вивести залежність кута ag від значення S:

|  |  |
| --- | --- |
|  | (2.12) |

Таким чином, для представлення сегмента у складі кінематичної схеми введемо для сегмента позначення Sb, положення якого залежить від повороту відносно осі Z на кут theta, phi і величини S:

|  |  |
| --- | --- |
| Sb(theta, phi, S)= Rz(theta)×Rz(-phi(S))×Ty(a1)×Rz(psi(S))×Segment(b) | (2.13) |

## Фізичні обмеження моделі

Для розрахунку положення двох сегментів використовується теорема косинусів, яка зв'язує сторони трикутника a, S і c, а так як зміна значення S використовується для управління кінематичної схеми, то необхідно враховувати обмеження на існування трикутника, що будь-яка сторона трикутника менша за суму двох інших сторін. Це обмеження може бути виражене двома предикатами:

|  |  |
| --- | --- |
| CheckMax(S) = S < a + c - 0.01  CheckMin(S) = S > a - c + 0.01 | (2.14) |

які обидва набувають значення true у тому випадку, коли значення S знаходиться в допустимих межах.

## Загальна тривимірна математична модель

До цих пір розв'язання задачі розглядалося в межах однієї площини. Для отримання загальної тривимірної моделі до повороту відносно осі Z додамо масштабування та повороти щодо осей X та Y. Всі перетворення, які виконуються для всіх сегментів, позначимо як Q:



тоді повна тривимірна модель набуде наступного вигляду:

|  |  |
| --- | --- |
| Sc(theta)= Rz(theta)×Segment(с)  Sa(theta, S)= Rz(theta)×Rz(-phi(S))×Segment(a)  Sb(theta, phi, S)= Rz(theta)×Rz(-phi(S))×Ty(a1)×Rz(psi(S))×Segment(b) | (2.16) |

а перетворення, які визначені в Q для ефективності можна зберегти, наприклад, у стеку матриць з подальшим відновленням у потрібний момент.

# ПРОГРАМНА РЕАЛІЗАЦІЯ КІНЕМАТИЧНОЇ СХЕМИ

## Початкові дані

Вихідні дані кінематичної схеми задані основними значеннями a, b, c які можуть бути представлені звичайними змінними:

public partial class RenderControl : OpenGL

{

public double a { get { return 0.6; } }

public double b { get { return 1.2; }}

public double c { get { return 0.54; } }

...

}

Аналогічним чином можуть бути реалізовані обмеження (2.14) для контролю за допустимими значеннями параметру S:

private double s = 0.2;

public double S

{

get { return s; }

set

{ if (value < (a + c - 0.01) && value > (a - c + 0.01)) s = value; }

}

крім цього, кути phi і psi у свою чергу теж є функціональними залежностями від значення S, що відображено у (2.9) и (2.12) і, відповідно, повинні мати заборону на операцію запису (і водночас переводимо радіани в градуси):

public double psi

{

get

{

return 180.0 / Math.PI \* (Math.PI - Math.Acos((a\*a + S\*S - c\*c) / (2\*S\*a)));

}

}

public double phi

{

get

{

return 180.0 / Math.PI \* Math.Acos((a\*a + c\*c - S\*S)/(2\*a\*c));

}

}

## Прив'язка системи координат до видимої області вікна

Для коректного відображення пропорцій елементів кінематичної схеми необхідно використовувати ізотропну систему координат, що може бути реалізовано, наприклад, в обробнику події, пов'язаної зі зміною розмірів вікна таким чином:

if (ClientSize.Width > ClientSize.Height)

{

int dx = (ClientSize.Width - ClientSize.Height) / 2;

glViewport(dx, 0, ClientSize.Height, ClientSize.Height);

}

else

{

int dy = (ClientSize.Height - ClientSize.Width) / 2;

glViewport(0, dy, ClientSize.Width, ClientSize.Width);

}

. . .

glOrtho(-sz, sz, -sz, sz, -sz \* 3, sz \* 3);

## Керування об'єктом моделювання та організація інтерфейсу

Для організації інтерфейсу з користувачем використовуємо клавіатуру та маніпулятор «миша». Керування системою координат та масштабом зв'яжемо з подіями маніпулятора «миша». Зафіксуємо натискання на ліву кнопку в логічній змінній MoveAxes та збережемо координати, в яких знаходився маніпулятор:

private void OnDown(object sender, MouseEventArgs e)

{

MoveAxes = (e.Button == MouseButtons.Left);

dx = e.X; dy = e.Y;

}

потім, якщо ліва кнопка не відпущена і маніпулятор виконує рух, зв'яжемо переміщення по горизонталі з кутом повороту навколо осі Y, а переміщення по вертикалі з кутом повороту відносно осі X:

private void OnMove(object sender, MouseEventArgs e)

{

if (MoveAxes)

{

ay += (e.X - dx) / 2.0;

ax += (e.Y - dy) / 2.0;

dx = e.X; dy = e.Y;

Invalidate();

}

}

після чого знову зберігаємо поточні координати маніпулятора та повідомляємо вікно про необхідність виконати перемалювання робочої області за допомогою методу Invalidate(). Виконуємо відстеження переміщення маніпулятора доти, доки ліва кнопка не буде відпущена:

private void OnUp(object sender, MouseEventArgs e)

{

MoveAxes = MoveAxes && (e.Button != MouseButtons.Left);

}

Для управління масштабуванням скористаємося колесом прокручування, використовуючи значення e.Delta як збільшення масштабу:

private void OnWheel(object sender, MouseEventArgs e)

{

m += e.Delta / 2000.0f;

Invalidate();

}

Так само відстежуємо необхідні події (натискання на клавіатурі), зв'язавши зміну кута обертання theta на 1 градус з клавішами «Left» і «Right», значення S зміняться на величину 0.01 клавішами «Up» і «Down», кут обертання aw зміняться на 1 градус клавішами «PageUp» «PageDown»:

private void OnKeys(object sender, PreviewKeyDownEventArgs e)

{

switch(e.KeyCode)

{

case Keys.Left : thetha -= 1; break;

case Keys.Right : thetha += 1; break;

case Keys.PageUp : aw -= 1; break;

case Keys.PageDown: aw += 1; break;

case Keys.Up : S += 0.01; break;

case Keys.Down : S -= 0.01; break;

}

Invalidate();

}

## Осі координат

Для підвищення загальної наочності та спрощення орієнтації елементів кінематичної схеми можна використовувати зображення осей координат та координатної сітки наступним чином:

public void DrawAxis()

{

glDisable(GL\_LIGHTING);

glDisable(GL\_LIGHT0);

glLineWidth(3);

glBegin(GL\_LINES);

glColor3d(0.4, 0.4, 0.4);

glVertex3d(0, 0, 0); glVertex3d(1.0, 0.0, 0.0);

glVertex3d(0, 0, 0); glVertex3d(0.0, 1.0, 0.0);

glVertex3d(0, 0, 0); glVertex3d(0.0, 0.0, 1.0);

glEnd();

glColor3d(1, 1, 1);

Text("+X", 1, 0, 0);

Text("+Y", 0, 1, 0);

Text("+Z", 0, 0, 1);

}

public void DrawGrid()

{

glLineWidth(1);

double size = 15;

glColor3d(0.7, 0.7, 0.7);

glBegin(GL\_LINES);

for (double i = -size; i <= size; i += size / 10)

{

glVertex3d(i, 0, -size);

glVertex3d(i, 0, size);

}

for (double j = -size; j <= size; j += size / 10)

{

glVertex3d(-size, 0, j);

glVertex3d(size, 0, j);

}

glEnd();

}

## Виведення сегмента

Для виведення сегмента в початковому стані відповідно до (2.6) використовуємо наступний програмний код, додавши значення r, g, b для управління кольором фігури, що виводиться:

private void Segment(double size, double r, double g, double b)

{

glColor3d(r, g, b);

glLineWidth(5);

glBegin(GL\_LINES);

glVertex3d(0, 0, 0);

glVertex3d(0, size, 0);

glEnd();

glLineWidth(1);

}

## Використання квадратичних об’єктів

Для виведення сегмента у вигляді квадратичного об’єкта (циліндра) використовуємо наступний програмний код:

private void Segment(double size, double r, double g, double b)

{

glColor3d(r, g, b);

IntPtr quadric = gluNewQuadric();

glPushMatrix();

glRotated(-90, 1, 0, 0);

gluCylinder(quadric, 0.05, 0.05, size, 16, 16);

glPopMatrix();

gluDeleteQuadric(quadric);

}

## Порядок виконання афінних перетворень

В результаті формування зображення в остаточному вигляді відбувається в методі, який відповідає за перерисовку області зображення:

private void OnRender(object sender, EventArgs e)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT | GL\_STENCIL\_BUFFER\_BIT);

glEnable(GL\_DEPTH\_TEST);

glLoadIdentity();

double sz = 2.5;

glOrtho(-sz, sz, -sz, sz, -sz \* 3, sz \* 3);

glRotated(ax, 1, 0, 0);

glRotated(ay, 0, 1, 0);

glScaled(m, m, m);

\_l.DrawGrid();

\_l.DrawAxis();

glRotated(aw, 0, -1, 0);

glRotated(thetha, 0, 0, -1);

Segment(c, 0.0, 1.0, 1.0);

glRotated(-phi, 0, 0, -1);

Segment(a, 1.0, 0.0, 1.0);

glTranslated(0, a, 0);

glRotated(psi, 0, 0, -1);

Segment(b, 1.0, 1.0, 0.0);

}

# РЕЗУЛЬТАТИ ВІЗУАЛІЗАЦІЇ КІНЕМАТИЧНОЇ СХЕМИ

У результаті виконання розробленого додатка на екран виведено маніпулятор, що відповідає заданій кінематичній схемі (див. рисунки 4.1–4.4):

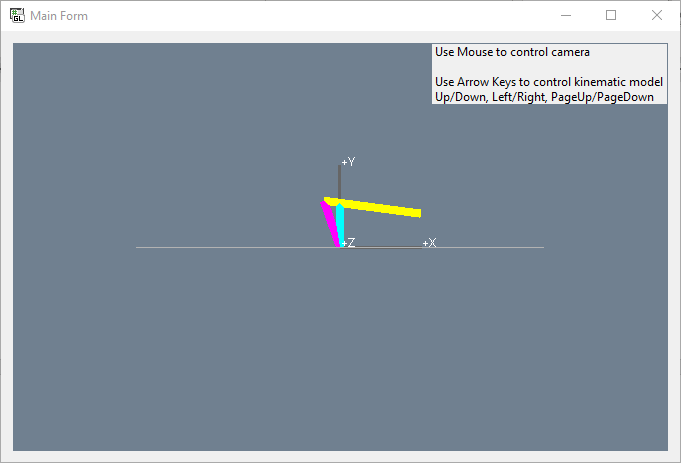


Рисунок 4.1 – Початковий стан кінематичної схеми після ініціалізації

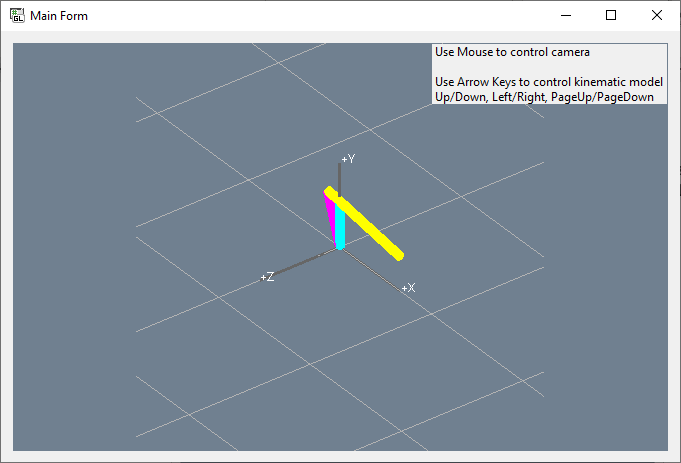


Рисунок 4.2 – Поворот системи координат, виконаний за допомогою маніпулятора “миша”

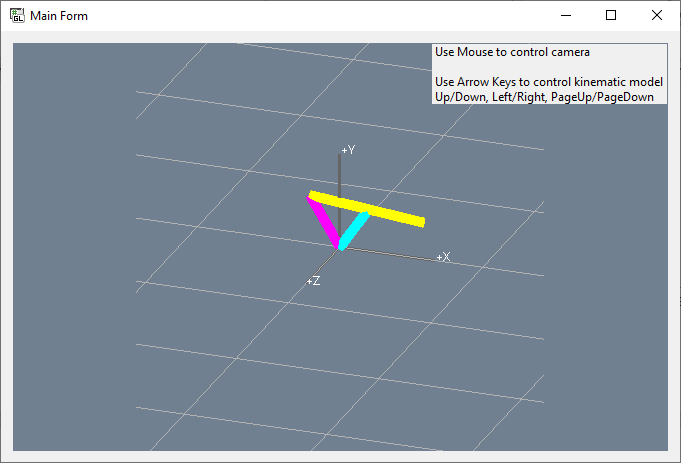


Рисунок 4.3 – Результат зміни параметрів θ та S

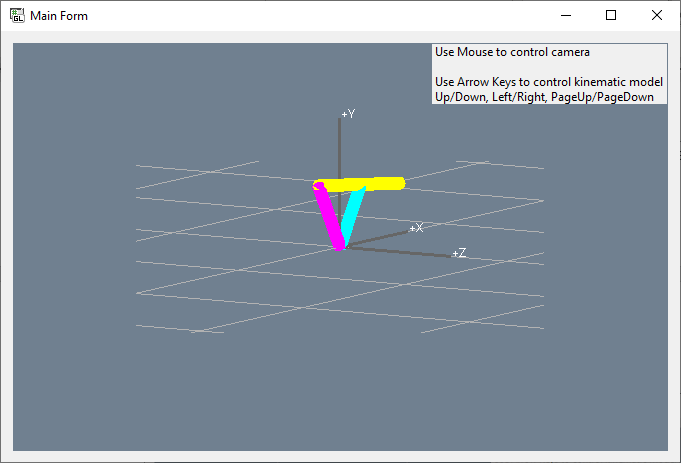


Рисунок 4.4 – Результат загального масштабування за допомогою колеса прокручування маніпулятора “миша”

# РЕАЛІЗАЦІЯ ТА КОНТРОЛЬ ВИМОГ ДО РОБОТИ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Складність | Вимоги до роботи | Бали | Оцінка |
| 1. | Базовий рівень | Реалізація програми двовимірної моделі маніпулятора відповідно до варіанта | 5 | + |
| 2. | Доопрацювання програми до тривимірної моделі (обертання точки спостереження, масштаб) | 1 | + |
| 3. | Управління моделлю та точкою спостереження маніпулятором «миша» та/або клавіатурою | 1 | + |
| 4. | Використання квадратичних примітивів для відображення кінематичної схеми | 3 | + |
| 5. | Використання освітлення та визначення матеріалів командою glColorMaterial (...) | 4 | – |
| 6. | Вміст звіту відповідає прикладу оформлення | 6 | + |
| 7. | Підвищений рівень | Використання ООП (розробка власних класів) | 1 | + |
| 8. | Використання текстур для елементів кінематичної схеми | 2 | – |
| 9. | Визначення матеріалів командою glMaterial (...), використання прозорості | 2 | – |
| 10. | Використання перспективної проекції для відображення моделі маніпулятора | 1 | – |
| 11. | Реалізація освітлення з тінню від моделі маніпулятора | 4 | – |

# ВИСНОВОК

У ході виконання підсумкової роботи було розібрано кінематичну схему шістнадцьотого варіанту, визначено афінні перетворення, виконано декомпозицію на сегменти; обраховано фізичні обмеження моделі; написано програмний код для керування кінематичної схеми у тривимірному просторі, реалізовано прив’язку системи координат до видимої області вікна, додано керування об’єктом моделювання за допомогою клавіатури та миши, виведено осі координат та координатну сітку, виведення сегментів за допомогою квадратичних об’єктів.

# ДОДАТОК Лістинг програми

**Код файлу RenderControl.cs:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Drawing;

using System.Drawing.Design;

using System.Drawing.Printing;

using System.Linq;

using System.Windows.Forms;

using static PR6\_Petrenko\_program.RenderControl;

namespace PR6\_Petrenko\_program

{

public partial class RenderControl : OpenGL

{

Layout \_l = new Layout();

public double a { get { return 0.6; } }

public double b { get { return 1.2; }}

public double c { get { return 0.54; } }

private double ay = 0;

private double ax = 0;

private double m = 1.0;

private bool MoveAxes = false;

private int dx;

private int dy;

private double thetha;

private double aw = 0;

private double s = 0.2;

public double S

{

get { return s; }

set

{ if (value < (a + c - 0.01) && value > (a - c + 0.01)) s = value; }

}

public double psi

{

get

{

return 180.0 / Math.PI \* (Math.PI - Math.Acos((a\*a + S\*S - c\*c) / (2\*S\*a)));

}

}

public double phi

{

get

{

return 180.0 / Math.PI \* Math.Acos((a\*a + c\*c - S\*S)/(2\*a\*c));

}

}

public RenderControl()

{

InitializeComponent();

\_l.Text = DrawText;

MouseWheel += OnWheel;

}

private void OnRender(object sender, EventArgs e)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT | GL\_STENCIL\_BUFFER\_BIT);

glEnable(GL\_DEPTH\_TEST);

glLoadIdentity();

if (ClientSize.Width > ClientSize.Height)

{

int dx = (ClientSize.Width - ClientSize.Height) / 2;

glViewport(dx, 0, ClientSize.Height, ClientSize.Height);

}

else

{

int dy = (ClientSize.Height - ClientSize.Width) / 2;

glViewport(0, dy, ClientSize.Width, ClientSize.Width);

}

double sz = 2.5;

glOrtho(-sz, sz, -sz, sz, -sz \* 3, sz \* 3);

glRotated(ax, 1, 0, 0);

glRotated(ay, 0, 1, 0);

glScaled(m, m, m);

\_l.DrawGrid();

\_l.DrawAxis();

glRotated(aw, 0, -1, 0);

glRotated(thetha, 0, 0, -1);

Segment(c, 0.0, 1.0, 1.0);

glRotated(-phi, 0, 0, -1);

Segment(a, 1.0, 0.0, 1.0);

glTranslated(0, a, 0);

glRotated(psi, 0, 0, -1);

Segment(b, 1.0, 1.0, 0.0);

}

//private void Segment(double size, double r, double g, double b)

//{

// glColor3d(r, g, b);

// glLineWidth(5);

// glBegin(GL\_LINES);

// glVertex3d(0, 0, 0);

// glVertex3d(0, size, 0);

// glEnd();

// glLineWidth(1);

//}

private void Segment(double size, double r, double g, double b)

{

glColor3d(r, g, b);

IntPtr quadric = gluNewQuadric();

glPushMatrix();

glRotated(-90, 1, 0, 0);

gluCylinder(quadric, 0.05, 0.05, size, 16, 16);

glPopMatrix();

gluDeleteQuadric(quadric);

}

private void OnDown(object sender, MouseEventArgs e)

{

MoveAxes = (e.Button == MouseButtons.Left);

dx = e.X; dy = e.Y;

}

private void OnUp(object sender, MouseEventArgs e)

{

MoveAxes = MoveAxes && (e.Button != MouseButtons.Left);

}

private void OnMove(object sender, MouseEventArgs e)

{

if (MoveAxes)

{

ay += (e.X - dx) / 2.0;

ax += (e.Y - dy) / 2.0;

dx = e.X; dy = e.Y;

Invalidate();

}

}

private void OnWheel(object sender, MouseEventArgs e)

{

m += e.Delta / 2000.0f;

Invalidate();

}

private void OnKeys(object sender, PreviewKeyDownEventArgs e)

{

switch(e.KeyCode)

{

case Keys.Left : thetha -= 1; break;

case Keys.Right : thetha += 1; break;

case Keys.PageUp : aw -= 1; break;

case Keys.PageDown: aw += 1; break;

case Keys.Up : S += 0.01; break;

case Keys.Down : S -= 0.01; break;

}

Invalidate();

}

}

}

**Код файлу Layout.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PR6\_Petrenko\_program

{

public partial class RenderControl

{

public class Layout

{

public delegate void outText(string s, double x, double y, double z);

public outText Text;

public void DrawAxis()

{

glDisable(GL\_LIGHTING);

glDisable(GL\_LIGHT0);

glLineWidth(3);

glBegin(GL\_LINES);

glColor3d(0.4, 0.4, 0.4);

//glColor3d(1, 0, 0);

glVertex3d(0, 0, 0); glVertex3d(1.0, 0.0, 0.0);

//glColor3d(0, 1, 0);

glVertex3d(0, 0, 0); glVertex3d(0.0, 1.0, 0.0);

//glColor3d(0, 0, 1);

glVertex3d(0, 0, 0); glVertex3d(0.0, 0.0, 1.0);

glEnd();

glColor3d(1, 1, 1);

Text("+X", 1, 0, 0);

Text("+Y", 0, 1, 0);

Text("+Z", 0, 0, 1);

}

public void DrawGrid()

{

glLineWidth(1);

double size = 15;

glColor3d(0.7, 0.7, 0.7);

glBegin(GL\_LINES);

for (double i = -size; i <= size; i += size / 10)

{

glVertex3d(i, 0, -size);

glVertex3d(i, 0, size);

}

for (double j = -size; j <= size; j += size / 10)

{

glVertex3d(-size, 0, j);

glVertex3d(size, 0, j);

}

glEnd();

}

}

}

}