

Матрице трансформације. Квадрат који се окреће

Матрице трансформације

За трансформацију дводимензионалних вектора се могу користити матрице димензије 2×2 :

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11} \cdot x + a_{12} \cdot y \\ a_{21} \cdot x + a_{22} \cdot y \end{bmatrix}.$$

Множење матрицом је линеарна трансформација вектора. Множењем матрицом димензије 2×2 се може дефинисати скалирање, смицање, ротација и рефлексја у дводимензионалном простору. Међутим, тако се не може дефинисати транслација. За померање је потребно

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_t \\ y_t \end{bmatrix}.$$

Уколико се дводимензионалне тачке представе помоћу $3D$ вектора $[x \ y \ 1]^T$ тада се добија да је

$$\begin{bmatrix} x_{new} \\ y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} \cdot x + m_{12} \cdot y + x_t \\ m_{21} \cdot x + m_{22} \cdot y + y_t \\ 1 \end{bmatrix}.$$

На исти начин се формирају матрице за трансформацију тродимензионалних вектора при чему се користе матрице 4×4 и вектори класе `Vector4D`.

OpenGL

До сада су прављене анимације које су се извршавале најбрже могуће и није контролисана брзина њиховог извршавања.

Методом

```
void glutTimerFunc(unsigned int msec, void (*func)(int value), value)
```

се може одложити извршавање неке функције. Када се жели одложено позивање неке методе, на месту регуларног позива (који би се одмах извршио) позва се функција `glutTimerFunc` којој се као други аргумент прослеђује баш та метода. Први параметар одређује минимално време у милисекундама које ће проћи пре позива прослеђене функције. Трећи параметар се може произвољно искористити.

Опис проблема

Креирати класе **Vector3D** и **Matrix3x3**. Класа *Vector3D* треба да представља тродимензионални вектор са методама за интензитет вектора, нормализацију, штампање, за секуларни и векторски производ, за пројекцију датог вектора на други вектор, као и предефинисаним операторима за сабирање, одузимање и множење. Класа *Matrix3x3* треба да садржи предефинисане операторе

за сабирање, одузимање и множење матрице матрицом, множење матрице вектором и множење матрице скаларом. Такође треба да садржи следеће методе:

- `void LoadTranslate(double dx, double dy)` – На основу задатих параметара поставља вредности матрице тако да се добије матрица translације за вектор $[dx \ dy]^T$.
- `void LoadScale(double sx, double sy)` – На основу задатих параметара поставља вредности матрице тако да се добије матрица скалирања и то по x оси sx пута, а по y оси sy пута.
- `void LoadRotateAboutOrigin(double alpha)` – На основу задатог угла поставља вредности матрице тако да се добије матрица ротације за угао α око осе која пролази кроз координатни почетак, а нормална је на раван xOy .

Задаци:

1. Направити анимацију у којој се квадрат окреће угаоном брзином од 20 степени по секунди око осе која пролази кроз доњу леву тачку квадрата и нормална је на раван xOy . Исцртавање вршити са 60 фрејмова по секунди.
2. Направити анимацију у којој се квадрат окреће угаоном брзином од 20 степени по секунди око осе која пролази кроз центар квадрата и нормална је на раван xOy . Исцртавање вршити са 60 фрејмова по секунди.

Решење

Пре свега је потребно креирати координате произвољног квадрата. Те координате се могу запамтити у глобалној променљивој. Како би се квадрат ротирао угаоном брзином од 20 степени по секунди, а исцртавање се врши 60 пута у секунди, угао ротације квадрата ће бити $20/60$ степени. Да би било могуће временски контролисати исцртавање коришћена је функција `glutTimerFunc`. У методи `timer` која се прослеђује функцији `glutTimerFunc` врши се трансформација тачака квадрата, као и позивање функције `glutPostRedisplay` да би се исцртао квадрат чија су темена ротирана. Такође, како би се наставила анимација, у функцији `timer` потребно је поново позивање методе `glutTimerFunc`. Да би се исцртавање вршило 60 фрејмова по секунди, први параметар при позиву функције `glutTimerFunc` имаће вредност $1000/60$.

У класи `Matrix3x3` направљена је функција за креирање матрице ротације око осе која пролази кроз координатни почетак, а нормална је на раван xOy . Како би се креирала матрица ротације која би се користила за ротирање задатог квадрата, потребно је креирати три матрице трансформације.

1. Нека је доња лева тачка квадрата A . Да би се извршило ротирање квадрата око тачке A , потребно је квадрат прво транслирати тако да се тачка A поклопи са координатним почетком (за вектор \overrightarrow{AO} добија се матрица translације $MTranslateAO$), потом ротирати квадрат за задати угао око координатног почетка ($MRotate$), и на крају транслирати квадрат тако да се тачка A врати у почетни положај (за вектор \overrightarrow{OA} добија се

матрица транслације $MTranslateOA$). Тако се коначна матрица ротације квадрата добија по формули:

$$MT = MTranslateOA * MRotate * MTranslateAO$$

```
void CreateTransformMatrix(Matrix3x3 &MT)
{
    Matrix3x3 MTranslateAO, MTranslateOA, MRotate;
    MTranslateAO.LoadTranslate(-1.0, -1.0);
    MTranslateOA.LoadTranslate(-1.0, 1.0);
    MRotate.RotateAroundOrigin(alpha);
    MT = MTranslateOA * MRotate * MTranslateAO;
}
```

2. Како је у овом примеру потребно ротирати квадрат око осе која пролази кроз центар квадрата, разлика у односу на претходни пример биће у транслирању квадрата. Тачније, прво треба извршити транслацију за вектор \overline{CO} (C центар квадрата) како би се центар квадрата нашао у координатном почетку, па након ротације око координатног почетка извршити транслацију за вектор \overline{OC} како би се центар квадрата вратио на почетни положај:

$$MT = MTranslateOC * MRotate * MTranslateCO$$

```
void CreateTransformMatrixCenter(Matrix3x3 &MT, Vector3D &C)
{
    Matrix3x3 MTranslateAO, MTranslateOA, MRotate;
    MTranslateAO.LoadTranslate(-C.X(), -C.Y());
    MTranslateOA.LoadTranslate(C.X(), C.Y());
    MRotate.RotateAroundOrigin(alpha);
    MT = MTranslateOA * MRotate * MTranslateAO;
}
```

Пример 1. Анимација – Квадрат који се ротира.

```
#include <GL/freeglut.h>
#include <vector>
#include <iostream>
#include "Vector2D.h"
#include "Vector3D.h"
#include "Matrix3x3.h"

using namespace std;

double w = 2.0; // angular velocity
int FPS = 60; // frames per sec
double alpha = w/FPS ;

Vector3D centar_kvadrata(1.5, 1.5, 0);
vector<Vector3D> kvadrat;
```

```

Matrix3x3 MT, MTcenter;

void CreateSquare()
{
    kvadrat.resize(4);
    kvadrat[0] = Vector3D(1.0, 1.0, 1.0);
    kvadrat[1] = Vector3D(2.0, 1.0, 1.0);
    kvadrat[2] = Vector3D(2.0, 2.0, 1.0);
    kvadrat[3] = Vector3D(1.0, 2.0, 1.0);
}

void CreateTransformMatrix(Matrix3x3 &MT)
{
    Matrix3x3 MTranslateAO, MTranslateOA, MRotate;
    MTranslateAO.LoadTranslate(-1.0, -1.0);
    MTranslateOA.LoadTranslate(-1.0, 1.0);
    MRotate.RotateAroundOrigin(alpha);
    MT = MTranslateOA * MRotate * MTranslateAO;
}

void CreateTransformMatrixCenter(Matrix3x3 &MT, Vector3D &C)
{
    Matrix3x3 MTranslateAO, MTranslateOA, MRotate;
    MTranslateAO.LoadTranslate(-C.X(), -C.Y());
    MTranslateOA.LoadTranslate(C.X(), C.Y());
    MRotate.RotateAroundOrigin(alpha);
    MT = MTranslateOA * MRotate * MTranslateAO;
}

void Transform(Matrix3x3 &M, vector<Vector3D> &v)
{
    for (int i = 0; i < v.size(); i++)
    {
        v[i] = M.Transform(v[i]);
    }
}

void DrawPolygon(vector<Vector3D> v)
{
    glBegin(GL_POLYGON);
    for (int i = 0; i < v.size(); i++)
    {
        glVertex3f(v[i].X(), v[i].Y(), v[i].Z());
    }

    glEnd();
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    DrawPolygon(kvadrat);
}

```

```

        glFlush();
    }

void timer(int v)
{
    Transform(MTcenter, kvadrat);
    glutTimerFunc(1000/FPS, timer, v);
    glutPostRedisplay();
}

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 3.0, 0.0, 3.0, -1.0, 1.0);
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("hello");
    init();

    CreateSquare();
    CreateTransformMatrix(MT);
    CreateTransformMatrixCenter(MTcenter, centar_kvadrata);

    glutDisplayFunc(display);
    glutTimerFunc(100, timer, 0);
    glutMainLoop();

    return 0;
}

```

Домаћи рад

Креирати класе `Vector4D` и `Matrix4x4`. `Vector4D` ће имати све операције које су наведене у претходном примеру за тродимензионални вектор. `Matrix4x4` треба да има све операције и методе наведене у претходном примеру за `Matrix3x3`. Узети у обзир да се ради са 4D векторима и сходно томе креирати матрице. Поред наведених, треба да постоје методе за ротацију вектора око X, Y и Z осе. Креиране класе биће коришћене на колоквијумима, тако да студенти имају слободу да додају методе које нису наведене, а сматрају да ће им бити потребне.