**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ВТ**

отчет

**по лабораторной работе №1**

**по дисциплине «Компьютерная графика»**

**Тема**: **Исследование математических методов представления и преобразования графических объектов на плоскости и в пространстве**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 6306 |  | Гордиенко М. Е. |
|  |  | Пустовойтова А. А. |
| Преподаватель |  | Матвеева И. В. |

Санкт-Петербург

2019

**Цель работы.**

Исследовать математические методы представления и преобразования графических объектов на плоскости и в пространстве.

**Задание.**

Сформировать отрезок, касательный к двум заданным окружностям с внешней стороны, определив предварительно координаты точек касания. Необходимо предусмотреть возможность редактирования положения и параметры окружностей.

**Теория.**

Под компьютерной графикой на плоскости, или двумерной компьютерной графикой понимают отображение на экране плоских, то есть двумерных объектов. В двумерной графике нет необходимости в операциях проецирования и наложения теней, так как объект плоский и расположен в одной плоскости – плоскости экрана. К геометрическим преобразованиям плоских объектов относятся сдвиг, поворот, масштабирование и отражение в плоскости экрана. Эти преобразования относятся к так называемым аффинным преобразованиям.

Напомним, что аффинной (общей декартовой) системой координат называется декартова система координат, в которой единицы масштаба (единицы отсчета) на координатных осях в общем случае различны.

Если в плоскости введены две аффинные системы координат, то преобразование, ставящее в соответствие точке P в одной системе координат точку P\* во второй системе координат и притом такую, которая во второй системе координат имеет такие же координаты, как и точка P в первой системе координат, называется аффинным. Аффинные преобразования сохраняют прямолинейность и параллельность линий, углы между ними, а также функциональные зависимости между параметрами геометрических фигур.

В компьютерной графике обычно применяются декартовы прямоугольные системы координат. В них точка на плоскости описывается парой координат x, y. При наличии в плоскости нескольких координатных систем перевод точки из одной системы в другую, в общем случае, описывается системой уравнений:

http://studepedia.org/img/baza1/52769543351591.files/image004.gif (2.1)

где x, y – координаты точки в «старой», а x\*, y\* – в «новой» системе координат;

tij, x0\*, y0\* – числа, связанные неравенством

http://studepedia.org/img/baza1/52769543351591.files/image005.gif http://studepedia.org/img/baza1/52769543351591.files/image007.gif

Приведенные выражения имеют и другой геометрический смысл. Они описывают новые координаты точки после выполнения над ней ряда геометрических преобразований в одной системе координат. Числа tij, x0\*, y0\* описывают параметры конкретных преобразований, но определить их значения для желаемого вида преобразований весьма затруднительно. В аффинных преобразованиях плоскости особую роль играют несколько важных частных случаев, для которых числовые коэффициенты уравнений перевода имеют ясный геометрический смысл. Это уже названные сдвиг, поворот, масштабирование и отражение.

Преобразование сдвига устанавливает соответствие между координатами точки в двух координатных системах, одна из которых сдвинута относительно другой на расстояние x0\* по горизонтали и y0\* по вертикали.

Применительно к компьютерной графике преобразование сдвига переводит координаты точки объекта из СКО (xOy) в СКН (в систему координат экрана x\*O\*y\*) при перемещении объекта:

http://studepedia.org/img/baza1/52769543351591.files/image009.gif

где x0\*, y0\* – координаты начала системы xOy в системе x\*O\*y\* .

Преобразование поворота (или – вращения) устанавливает соответствие между координатами точки объекта и экраном (СКН) при вращении объекта (без сдвига) относительно начала координат:

http://studepedia.org/img/baza1/52769543351591.files/image011.gif

где j – угол поворота СКО в СКН.

Другими словами, если центры СКО и СКН совпадают, то точка объекта, имеющая в СКО координаты x, y, при повороте объекта на угол j примет в СКН координаты x\*, y\* в соответствии с приведенными выражениями.

Преобразование масштабирования увеличивает или уменьшает размер изображения объекта в СКН по сравнению с исходным размером в СКО. При масштабировании назначается точка, относительно которой производится преобразование (неподвижная точка преобразования). Масштабирование относительно начала координат описывается уравнениями

http://studepedia.org/img/baza1/52769543351591.files/image013.gif

где k x , k y ¹ 0 – коэффициенты преобразования по координатным осям.

При k x, y > 1 происходит увеличение изображения, при k x, y < 1 – уменьшение, а при k x ¹ k y форма изображения искажается.

Преобразование отражения (симметрии) формирует в СКН изображение объекта, симметричное исходному. Отражение относительно оси, проходящей через начало координат СКН под углом a к оси абсцисс, описывается уравнениями:

http://studepedia.org/img/baza1/52769543351591.files/image015.gif

Геометрическая иллюстрация частных аффинных преобразований представлена на рисунке 2. Как видно, все они «привязаны» к началу СКН и потому являются частными вариантами общего случая, представленного выражениями (2.1).

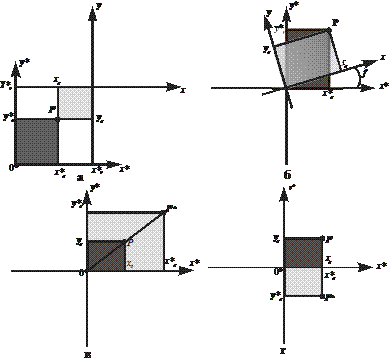


Рисунок 2 – Иллюстрация частных аффинных преобразований сдвига (а), поворота (б), масштабирования (в) и отражения (г)

Удобно записывать аффинные преобразования в матричной форме. Во-первых, можно компактно описывать сложные преобразования как сочетание (суперпозицию) простых. Во-вторых, в технических средствах компьютерной графики заложены возможности быстрого выполнения матричных операций (программно или аппаратно). Матричная запись в общем виде должна выглядеть так:

http://studepedia.org/img/baza1/52769543351591.files/image019.gif (2.2)

где T – матрица геометрического преобразования.

Для получения результирующих координат x\*, y\* нужно умножить матрицу-строку http://studepedia.org/img/baza1/52769543351591.files/image021.gif на матрицу T. Матрицу Tможно получить из системы уравнений (2.1), она может выглядеть только как транспонированная матрица коэффициентов этих выражений:



Однако приведенный вид матрицы невозможен для матричного умножения (2.2). Поскольку не совпадают размерности перемножаемых матриц, получить исходные выражения перевода их перемножением нельзя. Для установления соответствия размерностей в выражении (2.2) матрицы-строки http://studepedia.org/img/baza1/52769543351591.files/image025.gif должны иметь три элемента, то есть точка на плоскости должна описываться тремя координатами. Это возможно при использовании так называемых однородных координат.

Однородным представлением *n*-мерного объекта в математике, в общем случае, называют его представление в (*n*+1)-мерном пространстве, полученное добавлением еще одной координаты – скалярного множителя. Однородные координаты на плоскости определяются следующим образом. Пусть на плоскости в аффинной системе координат задана точка *P* с координатами (*x*, *y*). Однородными координатами этой точки называется любая тройка одновременно не равных нулю чисел [*w*1*w*2*w*3], связанных с координатами точки *P* соотношениями:

http://studepedia.org/img/baza1/52769543351591.files/image027.gif (2.3)

Безразмерное число *w*3 называется скалярным множителем.

Геометрически можно пояснить однородные координаты точки на плоскости, представив точку в некоторой условной пространственной системе координат (*x*,*y*,*w*), как это показано на рисунке 3.

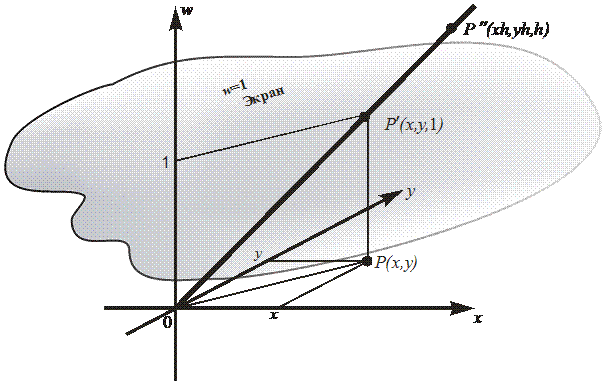
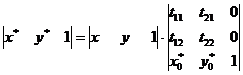


Рисунок 3 – Иллюстрация однородных координат точки на плоскости

Точку *P*(*x*,*y*) может представлять тройкой своих координат *w*1, *w*2, *w*3произвольная точка на прямой, соединяющей начало координат *О* (0, 0, 0) с точкой *P*¢ (*x*, *y*, 1). В частности, и точка *P*¢ однозначно определяет точку *P*, а также и любая точка *P*¢¢ c координатами (*xh*, *yh*, *h*), где *h* – скалярный множитель. В этом можно убедиться, воспользовавшись выражениями (2.3). Такое описание точки на плоскости называется в компьютерной графике описанием в однородных координатах. Представляют точку обычно так: (*x* : *y* : 1), то есть принимают *h*=1 , но применяют и общую форму: (*w*1 : *w*2: *w*3). Чтобы отличить однородное описание точки на плоскости (три координаты – *w*1, *w*2, *w*3) от привычного описания точки в декартовом пространстве (тоже три координаты – *x*, *y*, *z*), однородные координаты в описании точки разделяют не запятой, а двоеточием, например, *A*(*xA*:*yA*:1). Это правило не относится к матричному описанию точки, так как в математике знаки препинания между элементами матриц не ставятся, например, *KA* =|*xAyA* 1|.

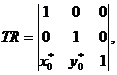
Теперь аффинное преобразование в общем виде будет выглядеть следующим образом:



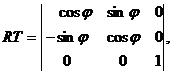
или *K*\* = *K* × *T*, (2.4)

где обозначения очевидны.

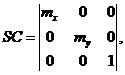
Перемножение матриц *K*и *T* даст два основных уравнения (2.1) перевода точки из СКО в СКН и верное числовое равенство 1=1. Следовательно, при помощи троек однородных координат и матриц третьего порядка можно описать любое частное аффинное преобразование. Так, матрица преобразований для сдвига (translation – перенос) принимает вид:



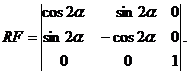
для вращения (rotation) –



для масштабирования (scaling) –



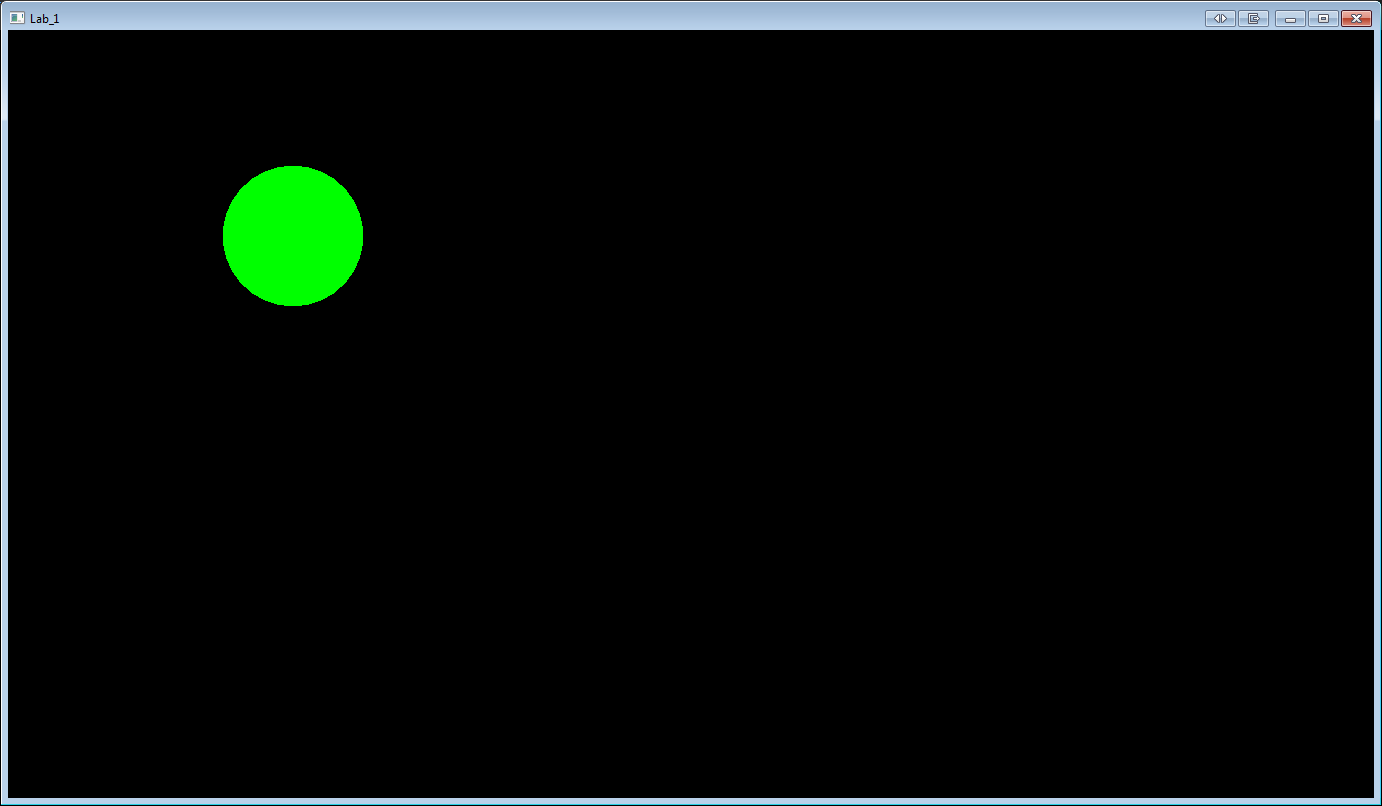
и для отражения (reflection) –



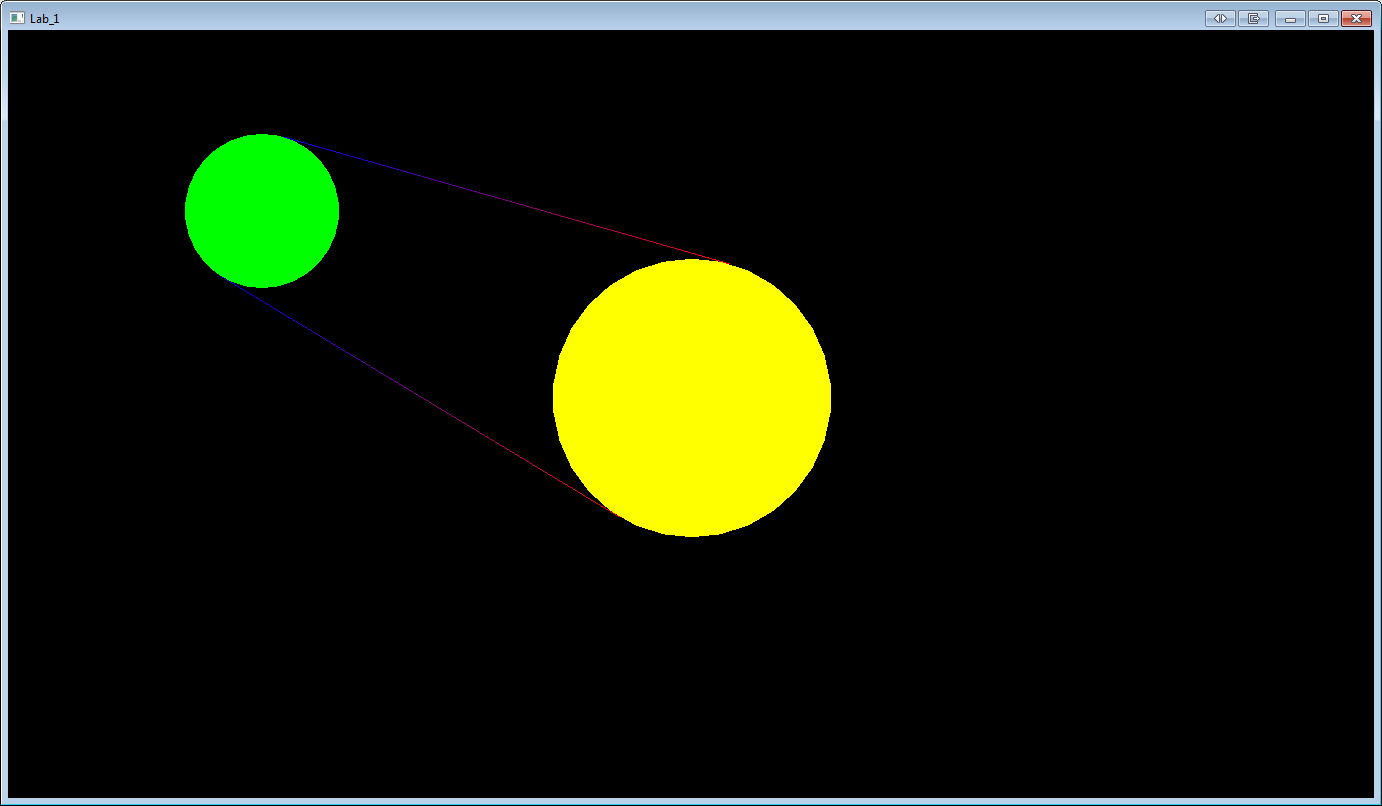
Отметим, что последняя матрица принимает более простой вид, если описывает отражение относительно оси x (α=0) или оси y (α=90°).

**Описание реализации.**

В открывшемся окне пользователем рисуется первая окружность (методом нажатия и растягивания).



Далее тем же способом рисуется вторая окружность, после чего программа автоматически дорисовывает к окружностям обе возможные внешние касательные.



**Вывод.**

В ходе выполнения работы были исследованы различные математические методы представления и преобразования графических объектов.

**Приложение.**

**Код программы.**

#include <SFML\Graphics.hpp>

#include <boost\numeric\ublas\matrix.hpp>

#include <boost\numeric\ublas\io.hpp>

using namespace boost::numeric::ublas;

int main()

{

sf::RenderWindow window(sf::VideoMode(1366, 768), "Lab\_1", sf::Style::Default);

//Круг 1(малый)

sf::CircleShape circle\_1(0.f);

sf::Vector2f circle\_1Pos;

float circle\_1Rad = 0.f;

circle\_1.setFillColor(sf::Color::Green);

bool circle\_1Ready = false;

//Кру 2(большой)

sf::CircleShape circle\_2(0.f);

sf::Vector2f circle\_2Pos;

float circle\_2Rad = 0.f;

circle\_2.setFillColor(sf::Color::Yellow);

bool circle\_2Ready = false;

//Касательные

sf::VertexArray tang\_1(sf::Lines, 2);

tang\_1[0].color = sf::Color::Blue;

tang\_1[1].color = sf::Color::Red;

sf::VertexArray tang\_2(sf::Lines, 2);

tang\_2[0].color = sf::Color::Blue;

tang\_2[1].color = sf::Color::Red;

bool tangReady = false;

while (window.isOpen())

{

sf::Event event;

while (window.pollEvent(event))

{

if (event.type == sf::Event::Closed || sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))

window.close();

}

if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {

if (!circle\_1Ready) { //Задание первого круга

circle\_1Pos.x = sf::Mouse::getPosition(window).x;

circle\_1Pos.y = sf::Mouse::getPosition(window).y;

circle\_1.setPosition(circle\_1Pos);

while (sf::Mouse::isButtonPressed(sf::Mouse::Left))

{

circle\_1.setPosition(circle\_1Pos.x - circle\_1Rad, circle\_1Pos.y - circle\_1Rad);

circle\_1Rad = sqrt(pow(abs(circle\_1.getPosition().x + circle\_1Rad - sf::Mouse::getPosition(window).x), 2) + pow(abs(circle\_1.getPosition().y + circle\_1Rad - sf::Mouse::getPosition(window).y), 2));

circle\_1.setRadius(circle\_1Rad);

window.clear();

window.draw(circle\_1);

window.display();

}

circle\_1Ready = true;

}

else if (!circle\_2Ready) { //Задание второго круга

circle\_2Pos.x = sf::Mouse::getPosition(window).x;

circle\_2Pos.y = sf::Mouse::getPosition(window).y;

circle\_2.setPosition(circle\_2Pos);

while (sf::Mouse::isButtonPressed(sf::Mouse::Left))

{

circle\_2.setPosition(circle\_2Pos.x - circle\_2Rad, circle\_2Pos.y - circle\_2Rad);

circle\_2Rad = sqrt(pow(abs(circle\_2.getPosition().x + circle\_2Rad - sf::Mouse::getPosition(window).x), 2) + pow(abs(circle\_2.getPosition().y + circle\_2Rad - sf::Mouse::getPosition(window).y), 2));

circle\_2.setRadius(circle\_2Rad);

window.clear();

window.draw(circle\_1);

window.draw(circle\_2);

window.display();

}

circle\_2Ready = true;

}

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Backspace) && (circle\_1Ready || circle\_2Ready)) { //Уничтожение нарисованных объектов

circle\_1.setRadius(0.f);

tang\_1[0].position = sf::Vector2f(0.f, 0.f);

tang\_1[1].position = sf::Vector2f(0.f, 0.f);

tang\_2[0].position = sf::Vector2f(0.f, 0.f);

tang\_2[1].position = sf::Vector2f(0.f, 0.f);

circle\_2.setRadius(0.f);

circle\_1Ready = false;

circle\_2Ready = false;

tangReady = false;

}

if (circle\_2Ready && !tangReady) { //Вычисление точек касания и отрисовка касательной

if (circle\_1Rad > circle\_2Rad) {

std::swap(circle\_1, circle\_2);

std::swap(circle\_1Pos, circle\_2Pos);

std::swap(circle\_1Rad, circle\_2Rad);

}

tang\_1[0].position = circle\_1Pos;

float tangLength = sqrt(pow(circle\_2Pos.x - circle\_1Pos.x, 2) + pow(circle\_2Pos.y - circle\_1Pos.y, 2) - pow(circle\_2Rad - circle\_1Rad, 2));

//Матрицы для операций поворота

vector<float> point1Vector(3);

matrix<float> moveMatrix(3, 3);

matrix<float> rotateMatrix(3, 3);

//Создание точки на отдалении в длину касательной от центра малой окружности

point1Vector(0) = circle\_1Pos.x + tangLength;

point1Vector(1) = circle\_1Pos.y;

point1Vector(2) = 1.f;

//Вспомогательная матрица для поворота

moveMatrix(0, 0) = 1; moveMatrix(0, 1) = 0; moveMatrix(0, 2) = 0;

moveMatrix(1, 0) = 0; moveMatrix(1, 1) = 1; moveMatrix(1, 2) = 0;

moveMatrix(2, 0) = -circle\_1Pos.x; moveMatrix(2, 1) = -circle\_1Pos.y; moveMatrix(2, 2) = 1;

//Основная матрица для поворота

rotateMatrix(0, 0) = (circle\_2Pos.x - circle\_1Pos.x) / sqrt(pow(circle\_2Pos.x - circle\_1Pos.x, 2) + pow(circle\_2Pos.y - circle\_1Pos.y, 2));

rotateMatrix(0, 1) = (circle\_2Pos.y - circle\_1Pos.y) / sqrt(pow(circle\_2Pos.x - circle\_1Pos.x, 2) + pow(circle\_2Pos.y - circle\_1Pos.y, 2));

rotateMatrix(0, 2) = 0;

//Вспомогательная матрица для поворота

rotateMatrix(1, 0) = -rotateMatrix(0, 1);

rotateMatrix(1, 1) = rotateMatrix(0, 0);

rotateMatrix(1, 2) = 0;

rotateMatrix(2, 0) = 0; rotateMatrix(2, 1) = 0; rotateMatrix(2, 2) = 1;

//Первые два перемножения матриц для поворота

point1Vector = prod(point1Vector, moveMatrix);

point1Vector = prod(point1Vector, rotateMatrix);

//На данном этапе точка лежит на расстоянии длины касательной от малой окружности в направлении центра большой окружности

//В сдвинутой системе координат

vector<float> point2Vector(3);

//Поворот точки и её копии до параллельности текущих отрезков необходимым касательным

point2Vector = point1Vector;

std::cout << point1Vector << std::endl;

tang\_1[1].position.x = point1Vector(0);

tang\_1[1].position.y = point1Vector(1);

rotateMatrix(0, 0) = tangLength / sqrt(pow(circle\_2Pos.x - circle\_1Pos.x, 2) + pow(circle\_2Pos.y - circle\_1Pos.y, 2));

rotateMatrix(0, 1) = (circle\_2Rad - circle\_1Rad) / sqrt(pow(circle\_2Pos.x - circle\_1Pos.x, 2) + pow(circle\_2Pos.y - circle\_1Pos.y, 2));

rotateMatrix(1, 0) = -rotateMatrix(0, 1);

rotateMatrix(1, 1) = rotateMatrix(0, 0);

point1Vector = prod(point1Vector, rotateMatrix);

rotateMatrix(0, 0) = tangLength / sqrt(pow(circle\_2Pos.x - circle\_1Pos.x, 2) + pow(circle\_2Pos.y - circle\_1Pos.y, 2));

rotateMatrix(0, 1) = -(circle\_2Rad - circle\_1Rad) / sqrt(pow(circle\_2Pos.x - circle\_1Pos.x, 2) + pow(circle\_2Pos.y - circle\_1Pos.y, 2));

rotateMatrix(1, 0) = -rotateMatrix(0, 1);

rotateMatrix(1, 1) = rotateMatrix(0, 0);

point2Vector = prod(point2Vector, rotateMatrix);

moveMatrix(2, 0) = circle\_1Pos.x;

moveMatrix(2, 1) = circle\_1Pos.y;

point1Vector = prod(point1Vector, moveMatrix);

point2Vector = prod(point2Vector, moveMatrix);

//"длина" и "ширина" касательных (длины их проекций на оси)

float tang1Width = point1Vector(0) - circle\_1Pos.x;

float tang1Height = point1Vector(1) - circle\_1Pos.y;

float tang2Width = point2Vector(0) - circle\_1Pos.x;

float tang2Height = point2Vector(1) - circle\_1Pos.y;

//Параллельный сдвиг текущих отрезков до положения искомых касательных

tang\_1[1].position.x = point1Vector(0) + (point1Vector(0) - circle\_2Pos.x) \* (circle\_1Rad / (circle\_2Rad - circle\_1Rad));

tang\_1[1].position.y = point1Vector(1) + (point1Vector(1) - circle\_2Pos.y) \* (circle\_1Rad / (circle\_2Rad - circle\_1Rad));

tang\_1[0].position.x = tang\_1[1].position.x - tang1Width;

tang\_1[0].position.y = tang\_1[1].position.y - tang1Height;

tang\_2[1].position.x = point2Vector(0) + (point2Vector(0) - circle\_2Pos.x) \* (circle\_1Rad / (circle\_2Rad - circle\_1Rad));

tang\_2[1].position.y = point2Vector(1) + (point2Vector(1) - circle\_2Pos.y) \* (circle\_1Rad / (circle\_2Rad - circle\_1Rad));

tang\_2[0].position.x = tang\_2[1].position.x - tang2Width;

tang\_2[0].position.y = tang\_2[1].position.y - tang2Height;

tangReady = true;

}

window.clear();

window.draw(circle\_1);

window.draw(circle\_2);

window.draw(tang\_1);

window.draw(tang\_2);

window.display();

}

return 0;

}