void AmiraSpatialGraph::applyTransformation()

{

//if(/\*!isIdentity && \*/ !transformationApplied)

{

// printTransformation();

transformationApplied = 1;

// std::flush(//std::cout << "Transforming " << vertices.size() << " vertices..." << std::endl);

std::vector< Vertex \* >::iterator vertexIt;

// int vertexCount = 1;

for(vertexIt = vertices.begin(); vertexIt != vertices.end(); ++vertexIt)

{

// //std::cout << "Transforming Vertex " << vertexCount << " of " << vertices.size() << std::endl;

// ++vertexCount;

double oldCoords[4], newCoords[4];

for(int ii = 0; ii < 3; ++ii)

{

oldCoords[ii] = (\*vertexIt)->coordinates[ii];

newCoords[ii] = 0;

}

oldCoords[3] = 1;

newCoords[3] = 1;

for(int ii = 0; ii < 3; ++ii)

for(int jj = 0; jj < 4; ++jj)

newCoords[ii] += transformation[ii][jj]\*oldCoords[jj];

for(int ii = 0; ii < 3; ++ii)

(\*vertexIt)->coordinates[ii] = newCoords[ii];

}

// std::flush(//std::cout << "Transforming " << edges.size() << " edges..." << std::endl);

std::vector< Edge \* >::iterator edgeIt;

// int edgeCount = 1;

for(edgeIt = edges.begin(); edgeIt != edges.end(); ++edgeIt)

{

// //std::cout << "Transforming Edge " << edgeCount << " of " << edges.size() << std::endl;

// //std::cout << (\*edgeIt)->edgePointCoordinates.size() << " points" << std::endl;

// ++edgeCount;

std::list< double \* >::iterator edgeListIt;

for(edgeListIt = (\*edgeIt)->edgePointCoordinates.begin(); edgeListIt != (\*edgeIt)->edgePointCoordinates.end(); ++edgeListIt)

{

double oldCoords[4], newCoords[4];

for(int ii = 0; ii < 3; ++ii)

{

oldCoords[ii] = (\*edgeListIt)[ii];

newCoords[ii] = 0;

}

oldCoords[3] = 1;

newCoords[3] = 1;

for(int ii = 0; ii < 3; ++ii)

for(int jj = 0; jj < 4; ++jj)

newCoords[ii] += transformation[ii][jj]\*oldCoords[jj];

for(int ii = 0; ii < 3; ++ii)

(\*edgeListIt)[ii] = newCoords[ii];

}

}

// std::flush(//std::cout << "done!" << std::endl);

}

};

void AmiraSpatialGraph::applyInverseTransformation(double \*\* inverse\_transformation)

{

if(true/\*!isIdentity && !inverseTransformationApplied\*/)

{

// printTransformation();

inverseTransformationApplied = 1;

// std::flush(//std::cout << "Transforming " << vertices.size() << " vertices..." << std::endl);

std::vector< Vertex \* >::iterator vertexIt;

// int vertexCount = 1;

for(vertexIt = vertices.begin(); vertexIt != vertices.end(); ++vertexIt)

{

// //std::cout << "Transforming Vertex " << vertexCount << " of " << vertices.size() << std::endl;

// ++vertexCount;

double oldCoords[4], newCoords[4];

for(int ii = 0; ii < 3; ++ii)

{

oldCoords[ii] = (\*vertexIt)->coordinates[ii];

newCoords[ii] = 0;

}

oldCoords[3] = 1;

newCoords[3] = 1;

for(int ii = 0; ii < 3; ++ii)

for(int jj = 0; jj < 4; ++jj)

newCoords[ii] += inverse\_transformation[ii][jj]\*oldCoords[jj];

for(int ii = 0; ii < 3; ++ii)

(\*vertexIt)->coordinates[ii] = newCoords[ii];

}

// std::flush(//std::cout << "Transforming " << edges.size() << " edges..." << std::endl);

std::vector< Edge \* >::iterator edgeIt;

// int edgeCount = 1;

for(edgeIt = edges.begin(); edgeIt != edges.end(); ++edgeIt)

{

// //std::cout << "Transforming Edge " << edgeCount << " of " << edges.size() << std::endl;

// //std::cout << (\*edgeIt)->edgePointCoordinates.size() << " points" << std::endl;

// ++edgeCount;

std::list< double \* >::iterator edgeListIt;

for(edgeListIt = (\*edgeIt)->edgePointCoordinates.begin(); edgeListIt != (\*edgeIt)->edgePointCoordinates.end(); ++edgeListIt)

{

double oldCoords[4], newCoords[4];

for(int ii = 0; ii < 3; ++ii)

{

oldCoords[ii] = (\*edgeListIt)[ii];

newCoords[ii] = 0;

}

oldCoords[3] = 1;

newCoords[3] = 1;

for(int ii = 0; ii < 3; ++ii)

for(int jj = 0; jj < 4; ++jj)

newCoords[ii] += inverse\_transformation[ii][jj]\*oldCoords[jj];

for(int ii = 0; ii < 3; ++ii)

(\*edgeListIt)[ii] = newCoords[ii];

}

}

// std::flush(//std::cout << "done!" << std::endl);

}

};

bool AxonDendriteProximityFinder::readAmiraTransformations()

{////std::cout<<"inputfilename: " << inputfilename <<std::endl;

std::ifstream inputStream(inputfilename);

if(!inputStream.fail())

{

const char \* letters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

const char \* numbers = "0123456789";

const char \* signs = "+-";

const char \* otherChars = ":;\'\"\\()[]{}!@#$%^&\_=|<>?";

const char \* whitespace = "\t ";

std::string currentLine;

unsigned int line = 0;

bool parameters = 1;

bool transform = 0;

// bool correctSection = 1;

// bool correctPrevSection = 0;

// int sectionID = 0;

unsigned int brackets = 0, transformBrackets = 0;

unsigned int currentIndex = 0;

while(!std::getline(inputStream, currentLine).eof() /\*&& line < 100\*/)

{

if(currentLine.size())

if(parameters && currentLine.find("TransformationMatrix ", 0) != std::string::npos)

{

// ////std::cout << "found correct section transform parameters!" << std::endl;

unsigned int count = 0;

std::string::size\_type loc1, loc2, loc3;

loc1 = currentLine.find\_first\_of(numbers, 0);

loc2 = currentLine.find\_first\_of(signs, 0);

if(loc2 != std::string::npos)

if(loc2 < loc1)

loc1 = loc2;

loc2 = currentLine.find\_first\_of(whitespace, loc1 + 1); //ignores last value: is always 1 anyways

while(loc2 != std::string::npos && count < 16)

{

char \* tmp1 = new char[loc2 - loc1];

currentLine.copy(tmp1, loc2 - loc1, loc1);

double ftmp1 = atof(tmp1);

sectionRotation[count%4][count/4]= ftmp1; // amira files are columns after each other

loc3 = loc2;

loc1 = currentLine.find\_first\_of(numbers, loc3);

loc2 = currentLine.find\_first\_of(signs, loc3);

if(loc2 != std::string::npos)

if(loc2 < loc1)

loc1 = loc2;

loc2 = currentLine.find\_first\_of(whitespace, loc1 + 1);

++count;

delete [] tmp1;

}

//sectionRotation = transformation;

//remove numeric artifacts from z-axis:

// for(int ii = 0; ii < 2; ++ii)

// {

// sectionRotation[2][ii] = 0;

// sectionRotation[ii][2] = 0;

// }

// sectionRotation[2][2] = 1;

for(int ii = 0; ii < 4; ++ii)

{

sectionTranslation[ii][3] = sectionRotation[ii][3];

sectionRotation[ii][3] = 0;

}

sectionRotation[3][3] = 1;

// sectionTranslation[2][3] += manual\_z\_scale[1]/\*\*manual\_z\_scale[0]\*/;

// ////std::cout<<"shift : "<<sectionTranslation[2][3]<<std::endl;

////std::cout << "translation matrix:" << std::endl;

for(int ii = 0; ii < 4; ++ii)

{

////std::cout << "[";

for(int jj = 0; jj < 4; ++jj)

{

//if(jj < 3)

////std::cout << sectionTranslation[ii][jj] << ",\t";

// else;

////std::cout << sectionTranslation[ii][jj];

}

////std::cout << "]" << std::endl;

}

////std::cout << "rotation matrix with scaling:" << std::endl;

for(int ii = 0; ii < 4; ++ii)

{

////std::cout << "[";

for(int jj = 0; jj < 4; ++jj)

{

//if(jj < 3)

////std::cout << sectionRotation[ii][jj] << ",\t";

//else;

////std::cout << sectionRotation[ii][jj];

}

////std::cout << "]" << std::endl;

}

double square\_scaling[2];

double scaling[4];

scaling[3] = 1;

for(int jj = 0; jj< 3; ++jj){

square\_scaling[jj] = sectionRotation[0][jj]\*sectionRotation[0][jj] + sectionRotation[1][jj]\*sectionRotation[1][jj] + sectionRotation[2][jj]\*sectionRotation[2][jj];

scaling[jj] = sqrt(square\_scaling[jj]);

////std::cout<<"scaling : "<<scaling[jj]<<std::endl;

}

//scaling[2] = manual\_z\_scale[0];

//////std::cout<<"scaling : "<<scaling[2]<<std::endl;

for(int ii = 0; ii < 3; ++ii)

{

for(int jj = 0; jj < 3; ++jj)

sectionRotation[ii][jj] = sectionRotation[ii][jj]/scaling[jj];

}

////std::cout << "rotation matrix without scaling:" << std::endl;

for(int ii = 0; ii < 4; ++ii)

{

////std::cout << "[";

for(int jj = 0; jj < 4; ++jj)

{

//if(jj < 3)

////std::cout << sectionRotation[ii][jj] << ",\t";

//else;

////std::cout << sectionRotation[ii][jj];

}

////std::cout << "]" << std::endl;

}

// double \*\* mInverse = new double \*[4];

// for(int ii = 0; ii < 4; ++ii)

// {

// mInverse[ii] = new double[4];

// for(int jj = 0; jj < 4; ++jj)

// mInverse[ii][jj] = 0;

// }

// for(int ii = 0; ii < 2; ++ii)

// for(int jj = 0; jj < 2; ++jj)

// mInverse[ii][jj] = sectionRotation[jj][ii];

// mInverse[0][3] = -1\*(sectionRotation[0][0]\*sectionTranslation[0][3] + sectionRotation[1][0]\*sectionTranslation[1][3]);

// mInverse[1][3] = -1\*(sectionRotation[0][1]\*sectionTranslation[0][3] + sectionRotation[1][1]\*sectionTranslation[1][3]);

// mInverse[2][3] = -1\*sectionTranslation[2][3];

// mInverse[3][3] = 1;

//

// this->inverse\_transformation = mInverse;

double \*\* mProduct = new double \*[4];

for(int ii = 0; ii < 4; ++ii)

{

mProduct[ii] = new double[4];

for(int jj = 0; jj < 4; ++jj)

mProduct[ii][jj] = 0;

}

for(int ii = 0; ii < 4; ++ii)

for(int jj = 0; jj < 4; ++jj)

for(int kk = 0; kk < 4; ++kk){

//////std::cout<<sectionTranslation[ii][kk]<<" \* "<<sectionRotation[kk][jj]<<" \* "<<scaling[jj]<<std::endl;

mProduct[ii][jj] += sectionTranslation[ii][kk]\*sectionRotation[kk][jj]\*scaling[jj];

}

this->transformation = mProduct;

////std::cout << "transformation matrix:" << std::endl;

for(int ii = 0; ii < 4; ++ii)

{

////std::cout << "[";

for(int jj = 0; jj < 4; ++jj)

{

//if(jj < 3)

////std::cout << transformation[ii][jj] << ",\t";

//else;

////std::cout << transformation[ii][jj];

}

////std::cout << "]" << std::endl;

}

// for(int ii = 0; ii < 2; ++ii)

// {

// transformation[2][ii] = 0;

// transformation[ii][2] = 0;

// }

// transformation[2][2] = 1;

double \*\* mInverse = new double \*[4];

for(int ii = 0; ii < 4; ++ii)

{

mInverse[ii] = new double[4];

for(int jj = 0; jj < 4; ++jj)

mInverse[ii][jj] = 0;

}

double a1 = sectionRotation[0][0]\*sectionRotation[1][1]\*sectionRotation[2][2];

double a2 = sectionRotation[0][1]\*sectionRotation[1][2]\*sectionRotation[2][0];

double a3 = sectionRotation[0][2]\*sectionRotation[1][0]\*sectionRotation[2][1];

double a4 = sectionRotation[2][0]\*sectionRotation[1][1]\*sectionRotation[0][2];

double a5 = sectionRotation[2][1]\*sectionRotation[1][2]\*sectionRotation[0][0];

double a6 = sectionRotation[2][2]\*sectionRotation[1][0]\*sectionRotation[0][1];

double det = (a1 + a2 + a3 -a4 -a5 -a6)\*scaling[0]\*scaling[1]\*scaling[2];

mInverse[0][0] = (sectionRotation[1][1]\*sectionRotation[2][2] - sectionRotation[1][2]\*sectionRotation[2][1])\*scaling[1]\*scaling[2]/det;

mInverse[0][1] = (sectionRotation[0][2]\*sectionRotation[2][1] - sectionRotation[0][1]\*sectionRotation[2][2])\*scaling[1]\*scaling[2]/det;

mInverse[0][2] = (sectionRotation[0][1]\*sectionRotation[1][2] - sectionRotation[0][2]\*sectionRotation[1][1])\*scaling[1]\*scaling[2]/det;

mInverse[1][0] = (sectionRotation[1][2]\*sectionRotation[2][0] - sectionRotation[1][0]\*sectionRotation[2][2])\*scaling[0]\*scaling[2]/det;

mInverse[1][1] = (sectionRotation[0][0]\*sectionRotation[2][2] - sectionRotation[0][2]\*sectionRotation[2][0])\*scaling[0]\*scaling[2]/det;

mInverse[1][2] = (sectionRotation[0][2]\*sectionRotation[1][0] - sectionRotation[0][0]\*sectionRotation[1][2])\*scaling[0]\*scaling[2]/det;

mInverse[2][0] = (sectionRotation[1][0]\*sectionRotation[2][1] - sectionRotation[1][1]\*sectionRotation[2][0])\*scaling[1]\*scaling[0]/det;

mInverse[2][1] = (sectionRotation[0][1]\*sectionRotation[2][0] - sectionRotation[0][0]\*sectionRotation[2][1])\*scaling[1]\*scaling[0]/det;

mInverse[2][2] = (sectionRotation[0][0]\*sectionRotation[1][1] - sectionRotation[0][1]\*sectionRotation[1][0])\*scaling[1]\*scaling[0]/det;

// mInverse[0][0] = sectionRotation[1][1]/scaling[1]\*sectionRotation[2][2]/scaling[2] - sectionRotation[1][2]/scaling[2]\*sectionRotation[2][1]/scaling[1];

// mInverse[0][1] = sectionRotation[0][2]/scaling[2]\*sectionRotation[2][1]/scaling[1] - sectionRotation[0][1]/scaling[1]\*sectionRotation[2][2]/scaling[2];

// mInverse[0][2] = sectionRotation[0][1]/scaling[1]\*sectionRotation[1][2]/scaling[2] - sectionRotation[0][2]/scaling[2]\*sectionRotation[1][1]/scaling[1];

// mInverse[1][0] = sectionRotation[1][2]/scaling[2]\*sectionRotation[2][0]/scaling[0] - sectionRotation[1][0]/scaling[0]\*sectionRotation[2][2]/scaling[2];

// mInverse[1][1] = sectionRotation[0][0]/scaling[0]\*sectionRotation[2][2]/scaling[2] - sectionRotation[0][2]/scaling[2]\*sectionRotation[2][0]/scaling[0];

// mInverse[1][2] = sectionRotation[0][2]/scaling[2]\*sectionRotation[1][0]/scaling[0] - sectionRotation[0][0]/scaling[0]\*sectionRotation[1][2]/scaling[2];

// mInverse[2][0] = sectionRotation[1][0]/scaling[0]\*sectionRotation[2][1]/scaling[1] - sectionRotation[1][1]/scaling[1]\*sectionRotation[2][0]/scaling[0];

// mInverse[2][1] = sectionRotation[0][1]/scaling[1]\*sectionRotation[2][0]/scaling[0] - sectionRotation[0][0]/scaling[0]\*sectionRotation[2][1]/scaling[1];

// mInverse[2][2] = sectionRotation[0][0]/scaling[0]\*sectionRotation[1][1]/scaling[1] - sectionRotation[0][1]/scaling[1]\*sectionRotation[1][0]/scaling[0];

//

// for(int ii = 0; ii < 2; ++ii)

// for(int jj = 0; jj < 2; ++jj)

// mInverse[ii][jj] = sectionRotation[jj][ii];

//scaling[2] = 1;

mInverse[0][3] = -1\*(sectionRotation[0][0]/scaling[0]\*sectionTranslation[0][3] + sectionRotation[1][0]/scaling[0]\*sectionTranslation[1][3] + sectionRotation[2][0]/scaling[0]\*sectionTranslation[2][3]);

mInverse[1][3] = -1\*(sectionRotation[0][1]/scaling[1]\*sectionTranslation[0][3] + sectionRotation[1][1]/scaling[1]\*sectionTranslation[1][3] + sectionRotation[2][1]/scaling[1]\*sectionTranslation[2][3]);

mInverse[2][3] = -1\*(sectionRotation[0][2]/scaling[2]\*sectionTranslation[0][3] + sectionRotation[1][2]/scaling[2]\*sectionTranslation[1][3] + sectionRotation[2][2]/scaling[2]\*sectionTranslation[2][3]);

mInverse[3][3] = 1;

this->inverse\_transformation = mInverse;

// mInverse[0][3] = -1\*(sectionTranslation[0][3]);

// mInverse[1][3] = -1\*(sectionTranslation[1][3]);

// mInverse[2][3] = -1\*sectionTranslation[2][3];

// mInverse[3][3] = 1;

//this->inverse\_transformation = mInverse;

////std::cout << "inverse transformation matrix:" << std::endl;

for(int ii = 0; ii < 4; ++ii)

{

////std::cout << "[";

for(int jj = 0; jj < 4; ++jj)

{

//if(jj < 3)

////std::cout << inverse\_transformation[ii][jj] << ",\t";

//else;

////std::cout << inverse\_transformation[ii][jj];

}

////std::cout << "]" << std::endl;

}

// for(int ii = 0; ii < 3; ++ii)

// {

//

// for(int jj = 0; jj < 3; ++jj){

//

// mIn

//

// }

// }

}

}

}

inputStream.close();

return 0;

}