**Código de conexão ao banco de dados PostGIS via Terralib 5**

// Examples

#include "DataAccessExamples.h"

// TerraLib

#include <terralib/dataaccess/datasource/DataSourceFactory.h>

#include <terralib/datatype.h>

// STL

#include <iostream>

#include <exception>

std::unique\_ptr<te::da::DataSource> GetPostGISConnection()

{

// let's give the minimal server connection information needed to connect to the database server

std::string aux, user, password, host, port, path, query;

std::string strURI = "pgsql://"; // The base of the URI

std::cout << "Informe o Host do servidor postGIS (ENTER para aceitar padrao \'localhost\'): ";

std::getline(std::cin, aux);

host = aux.empty() ? "localhost" : aux;

std::cout << "Informe o numero da porta para acessar o servidor postGIS (ENTER para aceitar padrao \'5432\'): ";

std::getline(std::cin, aux);

port = aux.empty() ? "5432" : aux;

std::cout << "Informe o usuario para acessar o servidor postGIS (ENTER para aceitar padrao \'postgres\'): ";

std::getline(std::cin, aux);

user = aux.empty() ? "postgres" : aux;

std::cout << "Informe a senha para acessar o servidor postGIS (ENTER para aceitar padrao \'postgresql\'): ";

std::getline(std::cin, aux);

password = aux.empty() ? "postgresql" : aux;

std::cout << "Informe o nome do Banco de Dados para conectar o servidor postGIS (ENTER para aceitar padrao \'postgis\_232\'): ";

std::getline(std::cin, aux);

path = aux.empty() ? "postgis\_232" : aux;

std::cout << "Informe o cliente enconding para conectar o servidor postGIS (ENTER para aceitar padrao \'UTF-8\'): ";

std::getline(std::cin, aux);

query = aux.empty() ? "&PG\_CLIENT\_ENCODING=" + te::core::CharEncoding::getEncodingName(te::core::EncodingType::UTF8) : aux;

std::cout << "Inform the Connection Time Out to connect to your postGIS server (ENTER para aceitar padrao \'4\'): ";

std::getline(std::cin, aux);

query += aux.empty() ? "&PG\_CONNECT\_TIMEOUT=4" : "&PG\_CONNECT\_TIMEOUT=" + aux;

strURI += user + ":";

strURI += password + "@";

strURI += host + ":";

strURI += port + "/";

strURI += path + "?";

strURI += query;

// create a data source using the data source factory

std::unique\_ptr<te::da::DataSource> ds = te::da::DataSourceFactory::make("POSTGIS", strURI);

try

{

ds->open();

}

catch (const std::exception& e)

{

std::cout << "Banco de Dados " << host << "/" << path << " não pode ser usado!\n Entre com os parâmetros corretos para a conexão!\n";

std::cout << "Erro: " << e.what() << std::endl;

ds.reset();

return ds;

}

catch (...)

{

std::cout << "Banco de dados " << host << "/" << path << " não pode ser usado!\n Entre com os parâmetros corretos para a conexão!\n";

ds.reset();

return ds;

}

std::cout << "Usando banco de dados " << host << "/" << path << std::endl;

ds->close();

return ds;

}

void PostGISExample()

{

try

{

std::unique\_ptr<te::da::DataSource> ds = GetPostGISConnection();

if (!ds.get())

return;

ds->open();

std::cout << "Banco de Dados esta aberto? " << std::boolalpha << ds->isOpened() << '\n';

std::cout << "Banco de Dados e valido? " << std::boolalpha << ds->isValid() << '\n';

// retrieve the data source capabilities and print it

std::cout << std::endl;

PrintDataSourceCapabilities(ds.get());

// printing some datasets from the datasource

std::vector<std::string> dsets = ds->getDataSetNames();

if (dsets.empty())

{

std::cout << "Banco de Dados nao tem tabelas.\n";

return;

}

// let's check one of them (or all)

std::cout << "\nEstes são " << dsets.size() << " os conjuntos de tabelas no banco de dados: \n";

for (size\_t i = 0; i<dsets.size(); ++i)

std::cout << '\t' << i + 1 << ':' << dsets[i] << std::endl;

// check point: retrieving the data from a dataset of the datasource

while (true)

{

std::cout << "\nSelecione um banco de dados de 1 até " << ds->getNumberOfDataSets() << " para ver os dados (0 para nenhum): ";

int n;

std::cin >> n;

if (n<1 || n>ds->getNumberOfDataSets())

break;

PrintDataSet(dsets[n - 1], ds->getDataSet(dsets[n - 1]).get());

}

system("PAUSE");

}

catch (const std::exception& e)

{

std::cout << std::endl << "An exception has occurred in the PostGIS Example: " << e.what() << std::endl;

}

catch (...)

{

std::cout << std::endl << "An unexpected exception has occurred in the PostGIS Example!" << std::endl;

}

}

**Código de cópia de dados para o PostGIS via Terralib 5**

// Examples

#include "DataAccessExamples.h"

// TerraLib

#include "../Config.h"

#include <terralib/dataaccess.h>

#include <terralib/geometry.h>

// STL

#include <iostream>

std::unique\_ptr<te::da::DataSource> GetPostGISConnection();

void CopyingData()

{

try

{

// let's take the input dataset from a shape file

std::string connInfo("file://");

std::string data\_dir = TERRALIB\_DATA\_DIR;

std::string aux("");

std::cout << "Informe a localização do shapefile (ENTER para aceitar o padrao \'" << (data\_dir + "/shape/segmentos.shp") << "\'): ";

std::getline (std::cin, aux);

if (!aux.empty())

connInfo += aux;

else

connInfo += data\_dir + "/shape/segmentos.shp";

std::unique\_ptr<te::da::DataSource> dsOrigin = te::da::DataSourceFactory::make("OGR", connInfo);

dsOrigin->open();

if (!dsOrigin->isValid())

{

std::cout << "Nao se pode acessar o shapefile.\n";

return;

}

// get a transactor to interact to the data source origin

std::auto\_ptr<te::da::DataSourceTransactor> tOrigin = dsOrigin->getTransactor();

std::vector<std::string> datasets = tOrigin->getDataSetNames();

std::auto\_ptr<te::da::DataSet> datasetOrigin = tOrigin->getDataSet(datasets[0]);

std::auto\_ptr<te::da::DataSetType> dtOrigin = tOrigin->getDataSetType(datasets[0]);

std::unique\_ptr<te::da::DataSource> dsDestination = GetPostGISConnection();

if (!dsDestination.get())

return;

dsDestination->open();

// get a transactor to interact to the data source

std::auto\_ptr<te::da::DataSourceTransactor> tDestination = dsDestination->getTransactor();

// create and save datasettype in the datasource destination

te::da::DataSetType\* newDataSet = static\_cast<te::da::DataSetType\*>(dtOrigin->clone());

newDataSet->setName("public.segmentos");

GetFirstGeomProperty(newDataSet)->setSRID(31983);

GetFirstGeomProperty(newDataSet)->setGeometryType(te::gm::GeometryType);

std::cout << std::endl << "Inicia a copia..." << std::endl;

std::map<std::string, std::string> options;

if (tDestination->dataSetExists("public.segmentos"))

{

tDestination->dropDataSet("public.segmentos");

}

tDestination->begin();

tDestination->createDataSet(newDataSet,options);

tDestination->add(newDataSet->getName(), datasetOrigin.get(),options);

tDestination->commit();

std::cout << std::endl << "Copia finilizada..." << std::endl;

}

catch(const std::exception& e)

{

std::cout << std::endl << "An exception has occurred in the Copy Example: " << e.what() << std::endl;

}

catch(...)

{

std::cout << std::endl << "An unexpected exception has occurred in the Copy Example!" << std::endl;

}

}

**Código TA\_Baatz\_Segmenter\_PostGIS**

#include "RPExamples.h"

// TerraLib

#include <terralib/raster.h>

#include <terralib/dataaccess.h>

#include <terralib/datatype.h>

#include <terralib/geometry.h>

#include <terralib/memory.h>

#include <terralib/rp.h>

#include <terralib/datatype.h>

// STL

#include <iostream>

#include <stdio.h>

#include <string>

#include <stdlib.h>

#include <cstdio>

#include <cstdio>

#include <memory>

#include <map>

#include <sstream>

#include <fstream>

#include <limits>

#include <libpq-fe.h>

bool getMaskRaster(const std::string& mask\_file\_name,

double geoWest, double geoNorth,

double geoEast, double geoSouth,

std::unique\_ptr< te::rst::Raster >& mask\_raster\_ptr)

{

if (mask\_file\_name.empty()) return false;

if (!(geoNorth >= geoSouth)) return false;

if (!(geoEast >= geoWest)) return false;

std::FILE\* file\_ptr = std::fopen(mask\_file\_name.c\_str(),

"rb");

if (file\_ptr == 0) return false;

char filetype[3];

unsigned int lines = 0;

unsigned int cols = 0;

char dummy\_char = 0;

if (6 != std::fscanf(file\_ptr, "%2s%1c%u%1c%u%1c",

filetype,

&dummy\_char,

&cols,

&dummy\_char,

&lines,

&dummy\_char)) {

std::fclose(file\_ptr);

std::cout << "Invalid file header";

return false;

}

if (std::string("P4") != filetype) return false;

/\* Allocating the raster \*/

std::vector< te::rst::BandProperty \* > bandsProps;

bandsProps.push\_back(new te::rst::BandProperty(0, te::dt::UCHAR\_TYPE));

te::gm::Envelope\* mbr = new te::gm::Envelope(geoWest, geoSouth, geoEast, geoNorth);

te::rst::Grid\* grid = new te::rst::Grid(cols, lines, mbr);

mask\_raster\_ptr.reset(te::rst::RasterFactory::make("MEM",

grid, bandsProps,

std::map< std::string, std::string >(), 0, 0));

/\* Reading data \*/

unsigned int col = 0;

unsigned int linedatasize = (cols / 8) +

((cols % 8) ? 1 : 0);

unsigned char\* linedata = new unsigned char[linedatasize];

unsigned int lineindex = 0;

te::rst::Raster& mask\_raster = (\*mask\_raster\_ptr);

double value = 0;

const unsigned char char\_1 = 1;

const unsigned char char\_2 = 2;

const unsigned char char\_4 = 4;

const unsigned char char\_8 = 8;

const unsigned char char\_16 = 16;

const unsigned char char\_32 = 32;

const unsigned char char\_64 = 64;

const unsigned char char\_128 = 128;

for (unsigned line = 0; line < lines; ++line)

{

if (1 == std::fread(linedata, linedatasize, 1, file\_ptr)) {

col = 0;

for (lineindex = 0; lineindex < linedatasize;

++lineindex) {

if (col < cols) {

value = ((linedata[lineindex] & char\_128) ? 0 : 255);

mask\_raster.setValue(col, line, value, 0);

++col;

}

if (col < cols) {

value = ((linedata[lineindex] & char\_64) ? 0 : 255);

mask\_raster.setValue(col, line, value, 0);

++col;

}

if (col < cols) {

value = ((linedata[lineindex] & char\_32) ? 0 : 255);

mask\_raster.setValue(col, line, value, 0);

++col;

}

if (col < cols) {

value = ((linedata[lineindex] & char\_16) ? 0 : 255);

mask\_raster.setValue(col, line, value, 0);

++col;

}

if (col < cols) {

value = ((linedata[lineindex] & char\_8) ? 0 : 255);

mask\_raster.setValue(col, line, value, 0);

++col;

}

if (col < cols) {

value = ((linedata[lineindex] & char\_4) ? 0 : 255);

mask\_raster.setValue(col, line, value, 0);

++col;

}

if (col < cols) {

value = ((linedata[lineindex] & char\_2) ? 0 : 255);

mask\_raster.setValue(col, line, value, 0);

++col;

}

if (col < cols) {

value = ((linedata[lineindex] & char\_1) ? 0 : 255);

mask\_raster.setValue(col, line, value, 0);

++col;

}

}

}

else {

std::fclose(file\_ptr);

delete[] linedata;

std::cout << "Error reading data";

return false;

}

}

std::fclose(file\_ptr);

delete[] linedata;

return true;

}

//-------------

// descricao dos campos

//

// class - class associated to the object.

// llx - lower left x coordinate of the object bounding box.

// lly - lower left y coordinate of the object bounding box.

// urx - upper right x coordinate of the object bounding box.

// ury - upper right y coordinate of the object bounding box.

// size - object size (in pixels).

// xCenter - x coordinate of the object centroid.

// yCenter - y coordinate of the object centroid.

// xGeoCenter - x geo-coordinate of the object centroid.

// yGeoCenter - y geo-coordinate of the object centroid.

// membership or p - confidence in the object with regard to its classification.

void saveToShp(std::vector<te::gm::Geometry\*> polygons, std::vector< double >& polygonsValues,

const std::string& shpBaseFileName, const te::rst::Grid& rasterGrid)

{

const double rasterPixelArea = rasterGrid.getResolutionX() \* rasterGrid.getResolutionY();

std::auto\_ptr<te::da::DataSetType> dataSetTypePtr1(new te::da::DataSetType(shpBaseFileName));

dataSetTypePtr1->add(new te::dt::SimpleProperty("value", te::dt::INT64\_TYPE, true));

dataSetTypePtr1->add(new te::dt::SimpleProperty("id", te::dt::INT64\_TYPE, true));

// inicio acréscimo campos

dataSetTypePtr1->add(new te::dt::SimpleProperty("llx", te::dt::DOUBLE\_TYPE, true));

dataSetTypePtr1->add(new te::dt::SimpleProperty("lly", te::dt::DOUBLE\_TYPE, true));

dataSetTypePtr1->add(new te::dt::SimpleProperty("urx", te::dt::DOUBLE\_TYPE, true));

dataSetTypePtr1->add(new te::dt::SimpleProperty("ury", te::dt::DOUBLE\_TYPE, true));

dataSetTypePtr1->add(new te::dt::SimpleProperty("size", te::dt::DOUBLE\_TYPE, true));

dataSetTypePtr1->add(new te::dt::SimpleProperty("xCenter", te::dt::DOUBLE\_TYPE, true));

dataSetTypePtr1->add(new te::dt::SimpleProperty("yCenter", te::dt::DOUBLE\_TYPE, true));

dataSetTypePtr1->add(new te::dt::SimpleProperty("xGeoCenter", te::dt::DOUBLE\_TYPE, true));

dataSetTypePtr1->add(new te::dt::SimpleProperty("yGeoCenter", te::dt::DOUBLE\_TYPE, true));

// fim acréscimo campos

dataSetTypePtr1->add(new te::gm::GeometryProperty("polygon", polygons[0]->getSRID(),

te::gm::PolygonType, true));

std::auto\_ptr<te::da::DataSetType> dataSetTypePtr2(new te::da::DataSetType(\*dataSetTypePtr1));

std::auto\_ptr< te::mem::DataSet > memDataSetPtr(new te::mem::DataSet(dataSetTypePtr1.get()));

double centroidCol = 0;

double centroidRow = 0;

for (unsigned int polygonsIdx = 0; polygonsIdx < polygons.size(); ++polygonsIdx)

{

te::mem::DataSetItem\* dsItemPtr = new te::mem::DataSetItem(memDataSetPtr.get());

rasterGrid.geoToGrid(polygons[polygonsIdx]->getCentroid().x,

polygons[polygonsIdx]->getCentroid().y, centroidCol, centroidRow);

dsItemPtr->setInt64(0, polygonsValues[polygonsIdx]);

dsItemPtr->setInt64(1, polygonsIdx);

//Novos campos

// llx

dsItemPtr->setDouble(2, polygons[polygonsIdx]->getMBR()->getLowerLeftX());

// lly

dsItemPtr->setDouble(3, polygons[polygonsIdx]->getMBR()->getLowerLeftY());

// urx

dsItemPtr->setDouble(4, polygons[polygonsIdx]->getMBR()->getUpperRightX());

// ury

dsItemPtr->setDouble(5, polygons[polygonsIdx]->getMBR()->getUpperRightY());

// size in pixels

dsItemPtr->setDouble(6, ((te::gm::Polygon\*)polygons[polygonsIdx])->getArea() / rasterPixelArea);

// xCenter

dsItemPtr->setDouble(7, centroidCol);

// yCenter

dsItemPtr->setDouble(8, centroidRow);

// xGeoCenter

dsItemPtr->setDouble(9, polygons[polygonsIdx]->getCentroid().x);

// yGeoCenter

dsItemPtr->setDouble(10, polygons[polygonsIdx]->getCentroid().y);

//Fim Novos campos

dsItemPtr->setGeometry(11, (te::gm::Geometry\*)polygons[polygonsIdx]->clone());

memDataSetPtr->add(dsItemPtr);

}

remove((shpBaseFileName + ".shx").c\_str());

remove((shpBaseFileName + ".shp").c\_str());

remove((shpBaseFileName + ".prj").c\_str());

remove((shpBaseFileName + ".dbf").c\_str());

std::string connInfo("file://" + shpBaseFileName + ".shp");

std::unique\_ptr<te::da::DataSource> dsOGR(te::da::DataSourceFactory::make("OGR", connInfo));

dsOGR->open();

memDataSetPtr->moveBeforeFirst();

te::da::Create(dsOGR.get(), dataSetTypePtr2.get(), memDataSetPtr.get());

dsOGR->close();

//return;

}

//--------inicio label image

void saveLabeledImageFile(const std::string& output\_image\_file\_name\_str,

const te::rst::Raster& raster, unsigned int raster\_channel)

{

// std::cout << "caminho " << output\_image\_file\_name\_str << std::endl;

// std::cout << "numero de bandas " << raster\_channel << std::endl;

// std::cout << "raster bandas " << raster.getNumberOfBands() << std::endl;

// std::cout << "nro linhas " << raster.getNumberOfRows() << std::endl;

// std::cout << "nro colunas " << raster.getNumberOfColumns() << std::endl;

if (!(raster.getNumberOfBands() > (int)raster\_channel)) throw;

unsigned int nlines = (unsigned int)raster.getNumberOfRows();

unsigned int ncols = (unsigned int)raster.getNumberOfColumns();

unsigned int curr\_line = 0;

unsigned int curr\_col = 0;

double curr\_value\_double = 0;

int curr\_value\_int = 0;

size\_t sizeofint = sizeof(int);

/\* Finding min and max raster values \*/

double max\_value\_dbl = std::numeric\_limits< double >::max() \* (-1.0);

double min\_value\_dbl = std::numeric\_limits< double >::max();

for (curr\_line = 0; curr\_line < nlines; ++curr\_line)

{

for (curr\_col = 0; curr\_col < ncols; ++curr\_col)

{

raster.getValue(curr\_col, curr\_line, curr\_value\_double,

raster\_channel);

if (curr\_value\_double > max\_value\_dbl) {

max\_value\_dbl = curr\_value\_double;

}

if (curr\_value\_double < min\_value\_dbl) {

min\_value\_dbl = curr\_value\_double;

}

}

}

unsigned int max\_value\_uint = (unsigned int)max\_value\_dbl;

unsigned int min\_value\_uint = (unsigned int)min\_value\_dbl;

/\* Creating file \*/

std::FILE\* file\_ptr = std::fopen(output\_image\_file\_name\_str.c\_str(),

"wb");

if (file\_ptr == 0)

{

std::cout << "Cannot create file";

system("PAUSE");

return;

}

/\* Writing header \*/

std::fprintf(file\_ptr, "F5\n");

#if BYTE\_ORDER == LITTLE\_ENDIAN

std::fprintf(file\_ptr, "L\n");

#elif BYTE\_ORDER == BIG\_ENDIAN

std::fprintf(file\_ptr, "B\n");

#else

#error "ERROR: Endianess format detection error"

#endif

std::fprintf(file\_ptr, "%u %u\n", ncols, nlines);

std::fprintf(file\_ptr, "%u %u\n", min\_value\_uint, max\_value\_uint);

/\* Writing image data \*/

for (curr\_line = 0; curr\_line < nlines; ++curr\_line) {

for (curr\_col = 0; curr\_col < ncols; ++curr\_col) {

raster.getValue(curr\_col, curr\_line, curr\_value\_double, raster\_channel);

curr\_value\_int = (int)curr\_value\_double;

if (1 != std::fwrite(&curr\_value\_int, sizeofint, 1, file\_ptr)) {

std::cout << "Error wrinting file";

std::fclose(file\_ptr);

return;

}

}

}

std::fclose(file\_ptr);

}

//--------Fim label image

//void Segmenter(){}

void Segmenter(char\* dbname, char\* username, char\* password, char\* hostname, char\* port, char\* tabela, char\* scale, char\* similarid, char\* color, char\* compact, char\* nrobandas, char\* pesobandas)

{

// -------------------------------------------------------------------------------------------------------------

// Region growing segmenter (with Baatz features) example

try

{

// Hora

time\_t rawtime;

struct tm \* timeinfo;

time(&rawtime);

timeinfo = localtime(&rawtime);

printf("Data inical atual do sistema: %s", asctime(timeinfo));

//std::cout << "Segmentacao Baatz - exemplo usando Raster no PostGIS" << std::endl << std::endl;

// open input raster

// Inicia abertura de imagem no banco e saída em disco

std::map<std::string, std::string> inputRasterInfo;

//std::cout << std::endl << "Ponto 1 - Abertura de imagem no banco" << std::endl;

std::stringstream ss;

ss << "PG:host=" << hostname << " port=" << port << " dbname='" << dbname << "' user='" << username << "' password='" << password << "' table=" << tabela << " schema='public' mode=2";

std::string s = ss.str();

inputRasterInfo["URI"] = s;

//std::cout << s;

boost::shared\_ptr< te::rst::Raster > inputRasterPointer(te::rst::RasterFactory::open(

inputRasterInfo));

if (inputRasterPointer.get() == 0) return;

/\* Access a raster datasource to create the output raster \*/

//std::cout << std::endl << "Ponto 2 - Define Saida em disco" << std::endl;

std::map<std::string, std::string> outputRasterInfo;

std::map<std::string, std::string> outputRasterInfoLabel;

outputRasterInfo["URI"] = TERRALIB\_DATA\_DIR "/geotiff/result\_img\_seg.tif";

outputRasterInfoLabel["URI"] = TERRALIB\_DATA\_DIR "/plm/result\_img\_seg\_plm.plm";

// Finaliza abertura de imagem no banco e saída em disco

// define segmentation parameters

// input parameters

te::rp::Segmenter::InputParameters algoInputParameters;

//abre parametro no disco

// algoInputParameters.m\_inputRasterPtr = rin;

//abre parametro no banco

algoInputParameters.m\_inputRasterPtr = inputRasterPointer.get();

/\*Tratamento do número de bandas\*/

std::string input = nrobandas;

std::istringstream nrob(input);

std::string token;

int qtd = 0;

while (std::getline(nrob, token, ',')) {

//std::cout << token << '\n';

int vl = atoi(token.c\_str());

//std::cout << vl << '\n';

algoInputParameters.m\_inputRasterBands.push\_back(vl);

qtd++;

}

// link specific parameters with chosen implementation

// strategy specific parameters (m\_minSegmentSize: size of the smallest segment to be created;

// m\_segmentsSimilarityThreshold: similarity between neighboring segments to merge them or not)

te::rp::SegmenterRegionGrowingBaatzStrategy::Parameters segparameters;

te::rp::SegmenterRegionGrowingBaatzStrategy::Parameters m\_minSegmentSize;

//-------------------------------------------

// segparameters.m\_minSegmentSize = 80; //Escala

std::stringstream st1;

st1 << scale;

int v1;

st1 >> v1;

segparameters.m\_minSegmentSize = v1;

//-------------------------------------------

// segparameters.m\_segmentsSimilarityThreshold = 0.03; //Similaridade

std::stringstream st2;

st2 << similarid;

double v2;

st2 >> v2;

segparameters.m\_segmentsSimilarityThreshold = v2;

//-------------------------------------------

// Peso das bandas -> iguais

// segparameters.m\_bandsWeights.clear();

//segparameters.m\_bandsWeights.resize(8, 0.125);

/\*Tratamento do peso de bandas\*/

std::string inputp = pesobandas;

std::istringstream pesb(inputp);

std::string tokenp;

//std::cout << inputp;

//float total = 0;

while (std::getline(pesb, tokenp, ',')) {

float vl = ::atof(tokenp.c\_str());

//total = total + vl;

//std::cout << vl << '\n';

segparameters.m\_bandsWeights.push\_back(vl);

}

//std::cout << total;

//exit;

//-------------------------------------------

// segparameters.m\_colorWeight = 0.7; // Cor

std::stringstream st3;

st3 << color;

double v3;

st3 >> v3;

segparameters.m\_colorWeight = v3;

//-------------------------------------------

segparameters.m\_compactnessWeight = 0.7; // Compacidade

std::stringstream st4;

st4 << compact;

double v4;

st4 >> v4;

segparameters.m\_compactnessWeight = v4;

//-------------------------------------------

algoInputParameters.m\_strategyName = "RegionGrowingBaatz";

algoInputParameters.setSegStrategyParams(segparameters);

//Processar a imagem em blocos (Emiliano) (Segmenter.h)

algoInputParameters.m\_enableThreadedProcessing = true;

algoInputParameters.m\_maxSegThreads = 0;

// blocagem

// algoInputParameters.m\_enableBlockProcessing = false;

algoInputParameters.m\_enableBlockProcessing = true;

algoInputParameters.m\_blocksOverlapPercent = 10;

algoInputParameters.m\_maxBlockSize = 0;

// output parameters

// the output can be a previously created raster (in this case,outputRasterInfo)

te::rp::Segmenter::OutputParameters algoOutputParameters;

//algoOutputParameters.m\_rInfo - As informações necessárias para criar o raster

algoOutputParameters.m\_rInfo = outputRasterInfo;

// algoOutputParameters.m\_rType = "GDAL";

// algoOutputParameters.m\_rType - Tipo de fonte de dados de raster de saída

algoOutputParameters.m\_rType = "EXPANSIBLE";

// execute the algorithm

te::rp::Segmenter seginstance;

if (!seginstance.initialize(algoInputParameters)) throw;

//std::cout << std::endl << "Ponto 3 - Segmentacao" << std::endl;

if (!seginstance.execute(algoOutputParameters)) throw;

//std::cout << std::endl << "Ponto 4 - Copiar raster para gerar vetor" << std::endl;

//\*algoOutputParameters.m\_outputRasterPtr - Um ponteiro do raster de saída gerado(imagem do rótulo)

if (!te::rp::Copy2DiskRaster(\*algoOutputParameters.m\_outputRasterPtr, outputRasterInfo["URI"])) throw;

//std::cout << std::endl << "Ponto 5 - Gerar Shapefile" << std::endl;

// Gera vetor

std::vector<te::gm::Geometry\*> polygons;

std::vector< double > polygonsValues;

algoOutputParameters.m\_outputRasterPtr->vectorize(polygons, 0, 0,

&polygonsValues);

//std::cout << std::endl << "Ponto 6 - salvar shapefile" << std::endl;

saveToShp(polygons, polygonsValues, "../../../\_exec\_seg/seg\_img",

\*algoOutputParameters.m\_outputRasterPtr->getGrid());

saveLabeledImageFile(outputRasterInfoLabel["URI"], \*algoOutputParameters.m\_outputRasterPtr, 0);

// Fim Gera vetor

// clean up

// Hora fim

time(&rawtime);

timeinfo = localtime(&rawtime);

printf("Data final atual do sistema: %s", asctime(timeinfo));

std::cout << "Pronto!" << std::endl << std::endl;

}

catch (const std::exception& e)

{

std::cout << std::endl << "Ocorreu uma exceção em Segmenter(): " << e.what() << std::endl;

}

catch (...)

{

std::cout << std::endl << "Ocorreu uma exceção inesperada em Segmenter()!" << std::endl;

}

return;

}

**Código da função de separação dos segmentos gerados no PostGIS e exportação para disco**

CREATE OR REPLACE FUNCTION public.fn\_tile2shp(

p\_tile\_id integer,

p\_tx\_tabela\_raster text,

p\_tx\_tabela\_vetor text)

RETURNS boolean AS

$BODY$

DECLARE

v\_srid\_raster integer;

c\_cursor\_segmentos record;

v\_command text;

BEGIN

execute 'create schema if not exists '||p\_tx\_tabela\_raster;

execute 'create table if not exists '||p\_tx\_tabela\_raster||'.export\_commands (

command text

)';

v\_command:= 'pgsql2shp -f C:/\_saida\_shapes/'||p\_tx\_tabela\_raster||'\_'||p\_tile\_id||' -h localhost -u postgres -P postgresql postgis\_232 '||p\_tx\_tabela\_raster||'.'||p\_tx\_tabela\_raster||'\_'||p\_tile\_id;

execute 'insert into '||p\_tx\_tabela\_raster||'.export\_commands (command)

values ($1)' using v\_command;

execute 'select st\_srid(rast) from '||p\_tx\_tabela\_raster||' where rid = $1' into v\_srid\_raster

using p\_tile\_id;

--raise exception '%', v\_srid\_raster;

execute 'drop table if exists '||p\_tx\_tabela\_raster||'.'||p\_tx\_tabela\_raster||'\_'||p\_tile\_id;

execute 'create table '||p\_tx\_tabela\_raster||'.'||p\_tx\_tabela\_raster||'\_'||p\_tile\_id||' as

select gid, st\_transform((st\_setsrid(geom,$1)) ,$1) as geom from '||p\_tx\_tabela\_vetor||' a

join '||p\_tx\_tabela\_raster||' b on st\_setsrid(geom,$1) && ST\_Envelope(rast) and st\_within(st\_setsrid(geom,$1),ST\_Envelope(rast))

where rid = $2

--and gid not in (select gid from '||p\_tx\_tabela\_raster||'.'||p\_tx\_tabela\_raster||'\_control)

union all

select a.gid, st\_transform(ST\_CollectionExtract(st\_intersection(st\_envelope(rast),st\_transform((st\_setsrid(geom,$1)) ,$1)),3),$1) as geom from '||p\_tx\_tabela\_vetor||' a

join '||p\_tx\_tabela\_raster||' b on st\_overlaps(st\_setsrid(geom,$1),ST\_Envelope(rast))

where rid = $2

'

using v\_srid\_raster, p\_tile\_id;

return true;

END;

$BODY$

LANGUAGE plpgsql VOLATILE

COST 100;

**Código do operador TA\_Baatz\_Segmenter\_PostGIS no InterImage**

<operator type=topdown class=gis name="TA\_Baatz\_Segmenter\_Postgis" cmd="C:/Terralib\_5.2.2/build/Debug/terralib\_example\_rp \"@dbname@\" \"@username@\" \"@password@\" \"@hostname@\" \"@port@\" \"@tabela@\" \"@scale@\" \"@similarity@\" \"@color@\" \"@compact@\" \"@nroband@\" \"@pesoband@\" " runglobal=false >

<attribute name=dbname label="a) Banco" value="postgis\_232">

<attribute name=username label="b) Usuario" type=string value="postgres">

<attribute name=password label="c) Senha" type=string value="postgresql">

<attribute name=hostname label="d) hostname" type=string value="localhost">

<attribute name=port label="e) Porta" type=integer value=5432>

<attribute name=tabela label="f) Tabela" type=string value=>

<attribute name=scale label="g) Escala" type=integer value=100>

<attribute name=similarity label="h) Similaridade" type=double value=0.03>

<attribute name=color label="i) Cor" type=double value=0.4>

<attribute name=compact label="j) Compacidade" type=double value=0.7>

<attribute name=nroband label="k) Nro. Bandas" type=double value=>

<attribute name=pesoband label="l) Pesos Bandas" type=double value=>

</operator>