#include "Python.h"

#include "arrayobject.h"

static PyObject \*diffuse(PyObject \*self, PyObject \*args);

// KH: Adapted Code from:

// KH: http://www.scipy.org/Cookbook/C\_Extensions/NumPy\_arrays

// KH: Adapted Code Begin

static char module\_docstring[] =

"This module provides an interface for calculating diffusion using C.";

static char diffusion\_docstring[] =

"Return the diffused array calculated from a 2D input array and obstacle array.";

static PyMethodDef module\_methods[] = {

{"diffuse", diffuse, METH\_VARARGS, diffusion\_docstring},

{NULL, NULL}

};

PyMODINIT\_FUNC initdiffuse(void) {

Py\_InitModule3("diffuse", module\_methods, module\_docstring);

/\* Load `numpy` functionality. \*/

import\_array();

}

/\* Create Carray from PyArray

\* Assumes PyArray is contiguous in memory.

\* Memory is allocated!

\*/

double \*\*doubleMatrixToCArrayPtrs(PyArrayObject \*arrayin, int numCols, int numRows) {

double \*\*c, \*a;

int i;

// Allocate a double \*vector (vec of pointers)

c = (double \*\*)malloc(numCols \* sizeof(double));

a = (double \*)arrayin->data; /\* pointer to arrayin data as double \*/

for (i = 0; i < numCols; i++) {

c[i] = a + i \* numRows;

}

return c;

}

int \*\*intMatrixToCArrayPtrs(PyArrayObject \*arrayin, int numCols, int numRows) {

int \*\*c, \*a;

int i;

// Allocate a double \*vector (vec of pointers)

c = (int \*\*)malloc(numCols \* sizeof(int));

a = (int \*)arrayin->data; /\* pointer to arrayin data as double \*/

for (i = 0; i < numCols; i++) {

c[i] = a + i \* numRows;

}

return c;

}

//KH: Adapted Code End

//KH: Original Code Begin

/\*

\* Diffuse a 2d numpy array using the given number of iterations, rate,

\* metric array and obstacle array.

v

\* Usage: newMetricArray = diffuse(int numIterations, float rate,

\* 2dNpAry metricArray, 2dNpAry obstacleAry)

\*/

static PyObject \*diffuse(PyObject \*self, PyObject \*args) {

PyArrayObject \*metricArray, \*obstacleArray, \*resultArray;

double rate;

double \*\*cMetricArray, \*\*cResultArray, \*\*cObstacleArray;

int numIterations, i, left, right, up, down, col, row, numCols, numRows;

int dimensions[2];

// parse metric and obstacle arrays and check return value

if (!PyArg\_ParseTuple(args, "idO!O!", &numIterations, &rate,

&PyArray\_Type, &metricArray,

&PyArray\_Type, &obstacleArray))

return NULL;

if (metricArray == NULL || obstacleArray == NULL)

return NULL;

numCols = metricArray->dimensions[0];

numRows = metricArray->dimensions[1];

dimensions[0] = numCols;

dimensions[1] = numRows;

resultArray = (PyArrayObject \*)PyArray\_FromDims(2, dimensions, NPY\_DOUBLE);

cMetricArray = doubleMatrixToCArrayPtrs(metricArray, numCols, numRows);

cObstacleArray = doubleMatrixToCArrayPtrs(obstacleArray, numCols, numRows);

cResultArray = doubleMatrixToCArrayPtrs(resultArray, numCols, numRows);

// copy metric array into result array

for (col = 0; col < numCols; col++) {

for (row = 0; row < numRows; row++) {

cResultArray[col][row] = cMetricArray[col][row];

}

}

for (i = 0; i < numIterations; i++) {

for (col = 0; col < numCols; col++) {

left = col - 1 < 0 ? numCols - 1 : col - 1;

right = col + 1 >= numCols ? 0 : col + 1;

for (row = 0; row < numRows; row++) {

// if this cell is not a goal or obstacle, diffuse

if (cObstacleArray[col][row] && cMetricArray[col][row] < 1.0) {

up = row - 1 < 0 ? numRows - 1 : row - 1;

down = row + 1 >= numRows ? 0 : row + 1;

// final diffusion value for col, row is the sum of neighbors times diffusion rate

cResultArray[col][row] = rate \* (cResultArray[left][row] +

cResultArray[right][row] +

cResultArray[col][up] +

cResultArray[col][down]);

}

}

}

// zero out obstacle cells in each iteration

for (col = 0; col < numCols; col++) {

for (row = 0; row < numRows; row++) {

if (!cObstacleArray[col][row]) {

cResultArray[col][row] = 0;

}

}

}

}

// free allocated memory

free((char\*)cMetricArray);

free((char\*)cObstacleArray);

free((char\*)cResultArray);

return PyArray\_Return(resultArray);

}

//KH: Original Code End