A Delaunay triangulation function in C

EricHufschmid

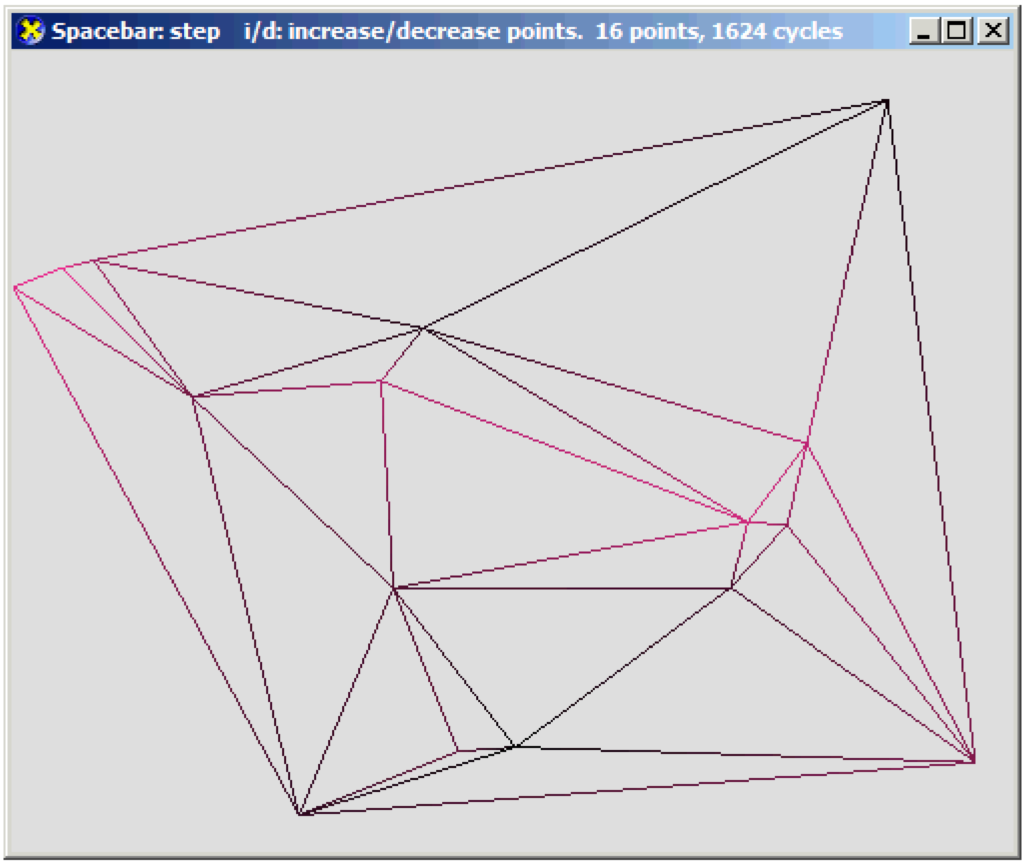
Rate me! ★★★★★ 4.92 (18 votes)

8 Sep 2014 CPOL

A C function to create a triangle index list

Download demo project - 40 KB

Download source - 11 KB


Spacebar: step   i/d: increase/decrease points. 16 points, 1624 cycles

Update, 6 September 2014:

When I created this project, I had just recently switched from the Borland compiler to Visual Studio 2012, and I made this triangulation project by modifying one of the DXUT samples in the DirectX SDK.

I have since discovered that Microsoft has some bizarre characteristics, such as including DXUT in the SDK, but advising us not to use it!

More importantly to you, I have recently used the triangulation function in a real project, and I was getting memory errors. Apparently, there are so many settings in the Microsoft compiler that the triangulation function works in some projects, but fails in others for reasons I cannot understand. In order to make the triangulation function work, I had to change this line :

```
xlm = ((simplex*) ( (char*)xlm + ((-1)) *simplex_size));
```

to this:

```
xlm = ((simplex*) ( (char*)xlm - simplex_size ));
```

Ken Clarkson wrote that function for the generic C compilers of the 1980s, and VS 2012 is not always interpreting `((-1)) *simplex_size)` the way he expected.

I updated the zip files with these changes.

Introduction

This function takes an array of 2D or 3D points, either integer values or floating-point values, and uses the Delaunay triangulation algorithm to create an index list that can be used directly in the DirectX or OpenGL functions that want triangle index lists.

It can triangulate several thousand points so quickly that you may be able to use this in real-time operations. It also has the option of setting the directions of the triangles to clockwise or anti-clockwise.

Background

I recently started learning DirectX, and I discovered that every shape has to be broken down into triangles, and all the triangles need to be indexed and put into a certain order, usually clockwise. I searched the Internet for C source code for a function that I could give an array of points, and which would give back a triangle index list, but all I could find were standalone programs that read and write text files, or functions in some other language. Furthermore, those programs didn't seem to care whether the triangles were clockwise or anti-clockwise, and the software provided by professors had restrictions.

Ken Clarkson posted the source code for his convex hull program without any strings attached, so I decided to extract his triangulation feature and make it into a C function. http://netlib.sandia.gov/voronoi/hull.html

Using the code

- The triangulation function is in one file: *Clarkson-Delaunay.cpp*
- It has only one header file: *Clarkson-Delaunay.h*
- You call only one function to use it: `BuildTriangleIndexList(...)`

I set all of the variables and functions to be "static" so that they don't interfere with the names in your own software.

To test the function, I wrote *test_triangulation.cpp* to create an array of random points, call `BuildTriangleIndexList()` to process those points into a triangle index list, and then display the triangles on the screen. Then it repeats the cycle.

It can triangulate the points so quickly that the triangles are a blur. You have to press the spacebar and step between each cycle in order to see the results.

I created the demo project with Visual Studio 2012. Since I am just starting to learn DirectX, I took Tutorial02 from the Microsoft DirectX SDK and added *Clarkson-Delaunay.cpp* and *test_triangulation.cpp* to it. The triangulation function is just a math function but, to satisfy the compiler, I had to put `#include "DXUT.h"` into it, even though it didn't need anything in the DirectX header files. I'm having trouble understanding the header files with Video Studio and DirectX. I can't even compile the samples from Microsoft without getting such warnings as "C4005: 'DXGI_STATUS_OCCLUDED': macro redefinition". My point in mentioning this is that you might have to change the "include" files in *Clarkson-Delaunay.cpp* in order for it to compile for you.

The triangulation function is: `WORD *BuildTriangleIndexList(void *, float, int *, int , int , int *);`.

It takes six arguments and returns a pointer to an array of triangle indices:

Hide Shrink ▲ Copy Code

```
WORD *BuildTriangleIndexList (
     void *pointList,          // INPUT, an array of either float or integer "points".
                                  // A "point" is either two values (an X and a Y),
                                  // or three values (XY and Z).
                                  // You must allocate memory and fill this array
                                  // with XY or XYZ points.
     float factor,             // INPUT, if pointList is a list of integers, set this
                                  // parameter to zero to let the function realize that
                                  // you are using integers. If you give this a value,
                                  // pointList will be interpreted as an array of
                                  // floating-point values.
                                  // Clarkson's function works on integers, so if
pointList
                                  // is an array of floating-point values, each value
will
                                  // be multiplied by this factor in order to convert
it to
                                  // an integer. Therefore, provide a factor that is
large
                                  // enough so that when the floating points are
converted,
                                  // you don't lose too many digits after the decimal
point,
                                  // unless you want to lose some digits.
                                  // Example: if you provide a factor of only 2.0, then
the
                                  // floating-point values 3.0001 and 3.499 will become
the
                                  // same integer value: 6
                                  // That would be acceptable if both of points are
supposed
```

```
                                    // to be the same, but otherwise it could cause
trouble.
     int numberOfInputPoints,   // INPUT, the number of points in the list,
                                   // not the number of bytes or values.
     int numDimensions,         // INPUT, 2 for X and Y points, or 3 for XY and Z points
     int clockwise              // There are three options:
                                   //     -1: put triangles in anti-clockwise order
                                   //      0: don't waste time ordering the triangles
                                   //      1: put triangles in clockwise order
     int *numTriangleVertices) // OUTPUT, this does not need to be initialized.
                                   //  BuildTriangleIndexList gives this a value.
```

The function does not need to be initialized or closed down. However, the calling function is responsible for releasing the return value with `free()`.

You are likely to want at least 4 variables to use it. In my demo program, I use these:

Hide Copy Code

```
WORD *triangleIndexList;   // OUTPUT, this does not need initialization
int *testPointsXY;         // INPUT, I fill this with random points for testing
int g_numTestPoints;       // INPUT, the number of points.
                           // I set this to 16 at the top of the
test_triangulation.cpp,
                           // and you can adjust its value as the program is running by
                           // pressing i or d on the keyboard
int numTriangleVertices;   // OUTPUT, this does not need initialization
```

Here is how I call it when I am passing floating-point values to it:

Hide Copy Code

```
triangleIndexList =  BuildTriangleIndexList(
          (void*)testPointsXY,       // The array of points
          (float)MAX_RAND,          // Multiplication factor. For testing: max random
value
          g_numTestPoints,          // The number of points
          2,                       // 2: the list is XY points, not XYZ points
          1,                      // 1, because I want the triangles clockwise
          &numTriangleVertices);
```

The return value is allocated by `BuildTriangleIndexList()`.

In the demo project zip file is *test_triangulation_integers.cpp*, which is the same as *test_triangulation.cpp*, except that it passes an array of integers to the triangulation function, in case you want to compare the difference between passing integer and floating-point arrays.

The return value from `BuildTriangleIndexList()` is an array of indices into the point list. You don't have to do anything to that array. Just put that variable and the number of vertices into the block of code that sets up the triangle index buffer, and in the statement that draws the triangles. For example, in my demo program:

Hide Copy Code

```
bd.Usage = D3D11_USAGE_DEFAULT;
bd.ByteWidth = sizeof( WORD ) * numTriangleVertices;     // <---- from the function
bd.BindFlags = D3D11_BIND_INDEX_BUFFER;
<..>
InitData.pSysMem =  triangleIndexList;                       // <---- same here
pd3dDevice->CreateBuffer( &bd, &InitData, &g_pTestVectorIndexBuffer );
<..>

g_pImmediateContext->DrawIndexed( numTriangleVertices, 0, 0);  // <---- same here
```

Also, free the return value:

Hide Copy Code

```
free ( triangleIndexList );
```

How to use the demo program:

The title bar of the window shows the number of points that are being created in each cycle, and the number of cycles that have been displayed.

Keyboard commands:

- **spacebar**: step between cycles rather than loop at full speed
- **i**: increase the number of points per cycle
- **d**: decrease the number of points per cycle
- **t**: toggle between displaying solid triangles or wireframe. When the triangles are solid, you can easily determine if they are all clockwise. If any are the wrong direction, they will not show.
- **ESC**: quit
- **Any other key**: loop as fast as possible.

Notes:

The output is a triangle index list, of 16-bit values, and each triangle requires three integers, one for each vertex. Therefore, to allocate memory for the output, the function needs to multiply the number of triangles by 3 WORDs:

Hide Copy Code

```
bytes needed for output array = (number_of_triangles * 3 * sizeof (WORD))
```

I don't know the number of triangles ahead of time, but in my tests, there were typically between 1.7 and 2 times as many triangles as input points, so I assume that there will never be more than 3 times as many triangles as input points. Therefore, I allocate memory for the output array like this:

Hide Copy Code

```
bytes needed for output array = (number_of_triangles * 3 * sizeof (WORD)) * 3
```

Which is why you see this at line 976:

Hide Copy Code

```
maxOutputEntries = numPointsProcessed * 3*3;
```

Clarkson mentions that his program suffers a performance loss of 2x to 3x because it uses exact arithmetic when possible. Clarkson designed his program to be accurate, which is wonderful when you need accuracy, but when calculating triangles for an image that appears on the screen only briefly, is such precision necessary? Somebody who understands his function might be able to modify it to accept a flag to either do a precise job, or a quick and sloppy job. Or, if his program is inherently precise, another program would have to be developed to do a quick and sloppy job.

Hide Copy Code

```
static void triangleList_out (int v0, int v1, int v2, int v3) {...}
```

When you give Clarkson's program an array of 2D points, (X and Y values), it gives back an array of three values for each triangle, an index to each vertex, which is what you expect. However, when you provide it with 3D points, (XY and Z values), it gives back a fourth value. I don't know what that fourth value is, so `triangleList_out()` is ignoring it.

1. Clarkson's program defines `MAXDIM` as **8**, which sets a maximum of 7 dimensions for the input points. Two dimensions is X and Y, and three dimensions is XYZ, but how does a point have more than 3 dimensions? I reduced `MAXDIM` to 4 because that variable was being used in calls to `malloc`, so that should reduce its memory requirements. If you are using points with four or more dimensions, you'll have to change `MAXDIM`.
2. Since Clarkson's program was designed to read and write files on disk, it didn't bother to figure out how many triangles there would be. However, as a function, it would be best to know the number of triangles so that the function needs only one call to malloc() when allocating memory for the triangle index list.
3. As you can see with the demo program, it can determine triangles extremely quickly, at least when the values are random and there are only up to several thousand points. It slows down noticeably when some of the points are in a grid format, such as, 1,2 1,3 1,4 1,5... 2,1, 2,2 2,3 etc.
4. Although not many people would attempt to modify Clarkson's logic, it is easy to modify the input and output functions. The output function is at line 1062:
5. The function returns a pointer to an array of 16-bit triangle indices. I don't know the limit of Clarkson's function is in regards to how many input points it will accept, but I set the return value to 16-bit integers because I assume nobody needs more than 64,000 triangles during one function call. If you want to return integers instead, you only need to do a search and replace in *Clarkson-Delaunay.cpp* to change `WORD` to `int`. The few references to `WORD` are from me, not from

Clarkson. He has some information on the abilities and limitations of his convex hull program here: http://netlib.sandia.gov/voronoi/hull.html.

Points of Interest

Ken Clarkson's source code has such cryptic variables and complex macros that it reminds me of the entries in the Obfuscated C Code Contest. His **STORAGE** macro is especially amusing. This is one macro:

Hide Shrink ▲ Copy Code

```
#define STORAGE(X)                          \
                                    \
size_t  X##_size;                           \
X   *X##_list = 0;                          \
                                    \
X *new_block_##X(int make_blocks)                   \
{   int i;                              \
    static  X *X##_block_table[max_blocks];         \
        X *xlm, *xbt;                       \
    static int num_##X##_blocks;                    \
/* long before; */                          \
    if (make_blocks) {                      \
/* DEBEXP(-10, num_##X##_blocks) */                 \
        assert(num_##X##_blocks<max_blocks);        \
/* before = _memfree(0);*/                      \
 DEB(0, before) DEBEXP(0, Nobj * ##X##_size)            \
                                    \
        xbt = X##_block_table[num_##X##_blocks++] = \
            (X*)malloc(Nobj * ##X##_size);          \
            memset(xbt,0,Nobj * ##X##_size);        \
/* DEB(0, after) DEBEXP(0, 8*_howbig((long*)xbt))*/     \
        if (!xbt) {                     \
            DEBEXP(-10,num_##X##_blocks)            \
/* memstats(2); */                          \
        }                           \
        assert(xbt);                        \
                                    \
        xlm = INCP(X,xbt,Nobj);                 \
        for (i=0;i<Nobj; i++) {                 \
            xlm = INCP(X,xlm,(-1));             \
            xlm->next = X##_list;               \
            X##_list = xlm;                 \
        }                           \
                                    \
        return X##_list;                    \
    };                              \
                                    \
    for (i=0; i<num_##X##_blocks; i++)              \
        free(X##_block_table[i]);               \
    num_##X##_blocks = 0;                       \
    X##_list = 0;                           \
    return 0;                           \
}                                   \
                                    \
void free_##X##_storage(void) {new_block_##X(0);}       \
```

Before you can decode that macro, you have to decode the three macros that are inside of it, namely: INCP, DEB, and DEBEXP:

Hide Copy Code

```
#define INCP(X,p,k) ((##X*) ( (char*)p + (k) * ##X##_size)) /* portability? */
#define DEB(ll,mes)  DEBS(ll) fprintf(DEBOUT,#mes "\n");fflush(DEBOUT); EDEBS
#define DEBEXP(ll,exp) DEBS(ll) fprintf(DEBOUT,#exp "=%G\n", (double) exp);
fflush(DEBOUT); EDEBS
```

But wait! Before you can decode those three macros, you have to decode the three macros that are within them, namely: DEBS, DEBOUT, and EDEBS:

Hide Copy Code

```
#define DEBS(qq)  {if (DEBUG>qq) {
#define EDEBS }}
#define DEBOUT DFILE
```

Finally, to complete the process, you need to know what DEBUG and DFILE are:

Hide Copy Code

```
#define DEBUG -7
extern FILE *DFILE;
```

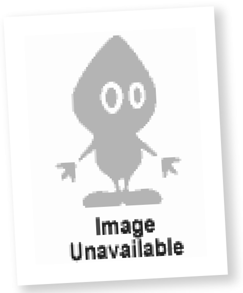
That type of macro makes me glad that I am a human rather than a C compiler!

In order to convert his program into a function, I had to figure out where memory was not being freed, and which variables needed initialization, but I could not make any sense of his macros, so I used my Borland 2007 compiler to expand his macros. That is why there are almost no macros in my variation of his software.

License

Share

About the Author



EricHufschmid

Software Developer Endpoint Software

United States

Follow this Member

I develop MillWrite, which is CAD/CAM and engraving software, for CNC routers, lasers, milling machines, and plasma cutters.

Comments and Discussions

Add a Comment or Question   Email Alerts   Search Comments

First Prev Next

3D triangulation

Jeremie84 2-Dec-16 14:51

Errors
Member 12575730 15-Jun-16 20:25

Re: Errors
EricHufschmid 16-Jun-16 22:56

Thanks a lot !
Member 11565670 30-Mar-15 8:53

Re: Thanks a lot !
sdelisle 3-Jun-15 13:46

Re: Thanks a lot !
EricHufschmid 16-Jun-16 23:08

Compute the Voronoi diagram
codep1 1-Oct-14 10:07

3D Point
Myagotin 6-Sep-14 14:34

Re: 3D Point
EricHufschmid 6-Sep-14 18:16

Re: 3D Point
Myagotin 7-Sep-14 8:06

Re: 3D Point
EricHufschmid 7-Sep-14 12:56

The BuildTriangleIndexList function is wrong
Gordius16 25-Jun-14 11:00

Re: The BuildTriangleIndexList function is wrong
EricHufschmid 25-Jun-14 12:41

Re: The BuildTriangleIndexList function is wrong
Gordius16 25-Jun-14 13:11

My vote of 5
Member 10893235 18-Jun-14 14:23

input and output format.
Member 10713525 21-May-14 5:55

Re: input and output format.
EricHufschmid 25-Jun-14 12:28

My vote of 5
SanCHEESE 10-Mar-14 4:24

format of the input point array
Member 8027114 9-Aug-13 13:40

Re: format of the input point array
EricHufschmid 25-Jun-14 11:50

license of Clarkson's work?
Member 10124906 25-Jun-13 2:37

Re: license of Clarkson's work?
EricHufschmid 25-Jun-13 13:05

My vote of 5
Mihai MOGA 14-Jun-13 2:45

My vote of 5
ed welch 10-Jun-13 6:16

when change NUM_DIMENSIONS to 3, Exception occurs when call BuildTriangleIndexList
solovelyl 18-May-13 1:08

Refresh 1 2 Next »

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.