Gridding the Earth for Climate Models

John Burkardt
Department of Scientific Computing
Florida State University
...
Trinity University
Mathematics Department Colloquium
20 February 2013
...
https://people.sc.fsu.edu/~jburkardt/presentations/...
sphere_grid_2013_trinity.pdf

The Spherical Mesh

- **Planar Meshes**
- The STRIPACK Sphere Mesh
- The SCVT Sphere Mesh
- Examples

PLANE: Analysis of PDE's

A fundamental problem of analysis is the study of the partial differential equations used to model the physical processes that we observe in nature.

We typically confined such problems to a particular bounded region or surface, and a limited time.

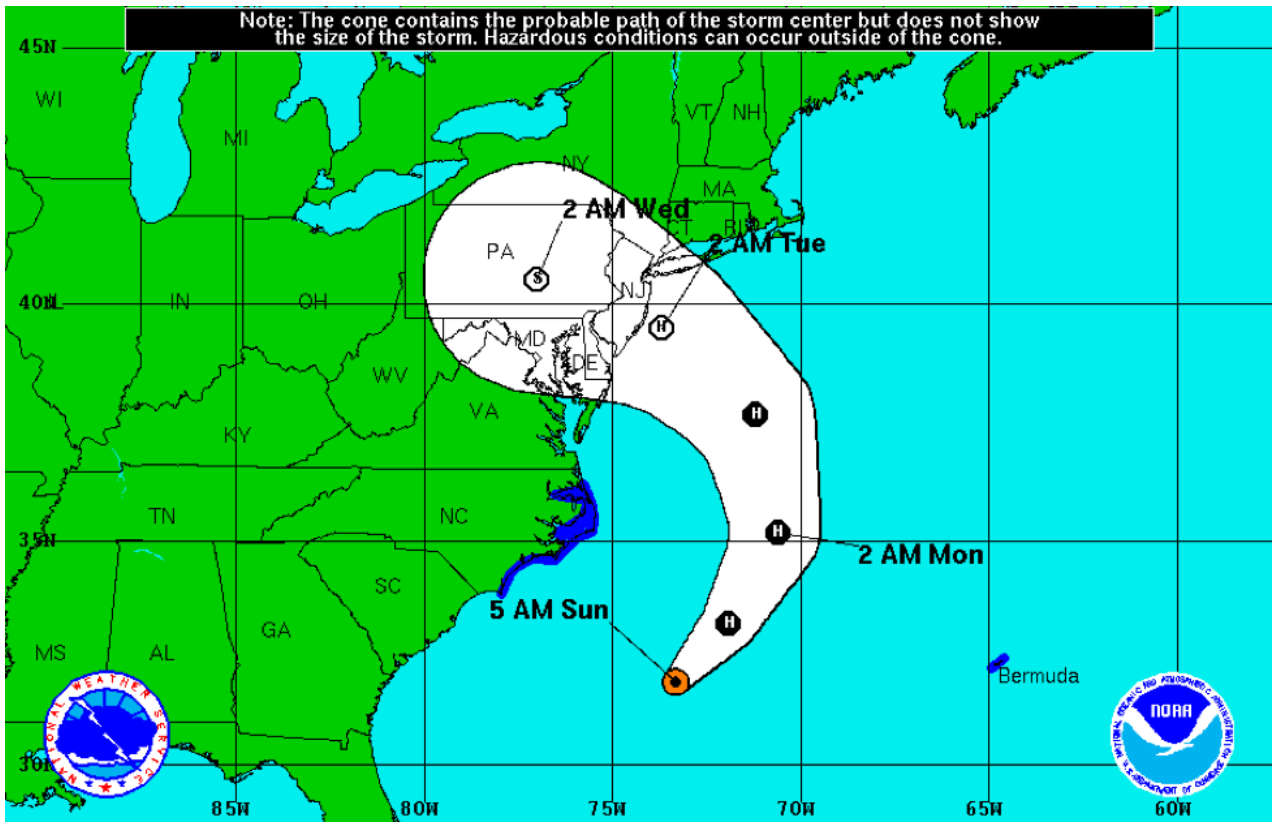
Answers to such problems, even approximate ones, can be of huge importance.

Such processes include coolant flow in a nuclear reactor, the design and analysis of airplane wings, and weather prediction.

PLANE: Hurricane's Are PDE's

In some sense, figuring out which way Hurricane Sandy is going to move, and how much damage it will cause, is just a really complicated homework problem.

But it's tremendously important to get an accurate and timely answer!



PLANE: Discretized Geometry

Typically, the partial differential equations describe how quantities such as mass, heat, velocity, pressure, electric potential or magnetic force vary in time over a specified region.

For special problems and regions, we might be able to determine a formula for the solution. But in general, we can only hope to construct approximations.

When such problems are handled on a computer, the geometry of the region is an important issue. A standard technique is to sprinkle the region and its boundary with many sample points, at which a solution is to be computed.

After discretizing the geometry, related techniques are used to approximate the differential operations that appear in the problem definition, and to solve the resulting system of differential or algebraic equations for the aproximate solution.

We will not proceed to the differential equation step, but instead, will focus on the issues involved in dealing with the **problem geometry**.

PLANE: Points, Areas, Connections

The unit square is the most popular "typical geometry" for describing physical problems.

Supposing we have a problem involving the transport of a scalar quantity through the region, carried along on some moving medium. How would we dissect our geometry?

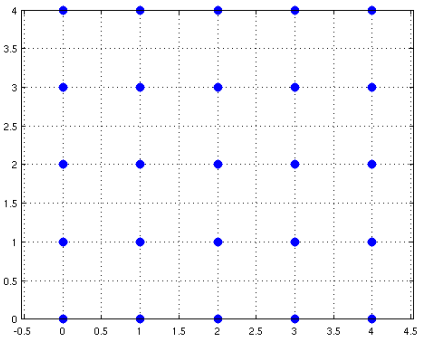
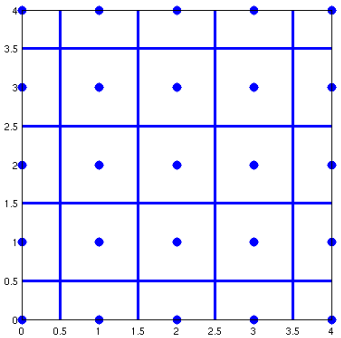
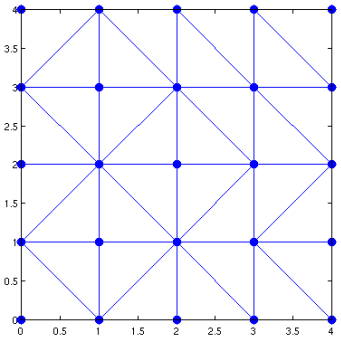
Because there will be boundary conditions, we will want sample points along the boundary lines, to register the inflow and outflow of the quantity. The interior of the region will need a set of points at which the unknown quanitity is to be computed.

When modeling the differential equation, we may need to know how the quantity varies over space. If we look for the value at the nearest sample point, we are implicitly dividing the region into subregions, for which the sample points are representative.

The partial differential equation describes, essentially, how the quantity changes locally, over infinitesmal distances and times. To approximate this measurement of change in space, each sample point must be somehow related to all its neighbors.

PLANE: The Unit Square

If we're on a unit square, it's quite easy to choose a uniform placement of sample points, to dissect the region into subregions, and to determine the connections between sample points.

- The sample points or **nodes** tell us where the values are known.
- The subregions or **elements** tell us over what extent that value is dominant.
- The connections or **triangulations** tell us what elements can "trade" data.

PLANE: General Geometry Points

Since our airplanes, cars, and countries aren't well modeled by unit squares, what happens when we try to set up and approximate a PDE system on a general region?

Dealing with the geometry of a general region, by itself, can absorb an overwhelming portion of the problem difficulty.

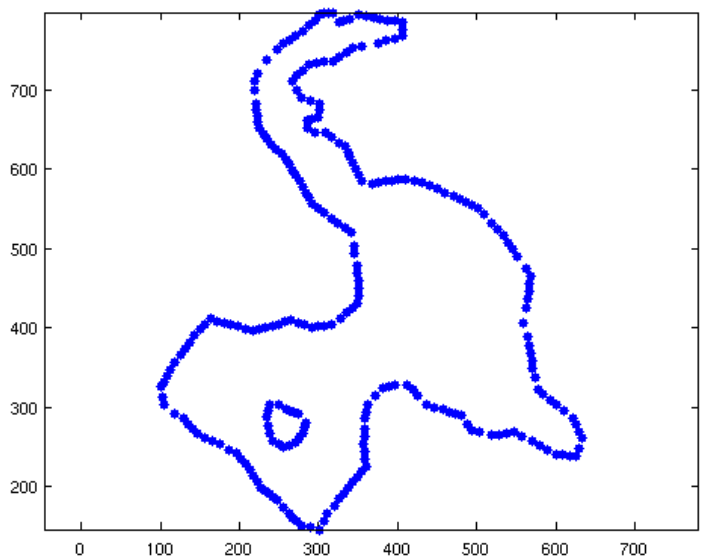
A curved boundary, or internal obstacles or holes, can be a significant feature that must be approximated well.

If the region includes a long and thin portion, then a volume-wise uniform distribution of points will severely undersample the thin region.

We need an **automatic method** to select well-distributed nodes, to set up the elements, and establish the triangulation.

PLANE: General Geometry

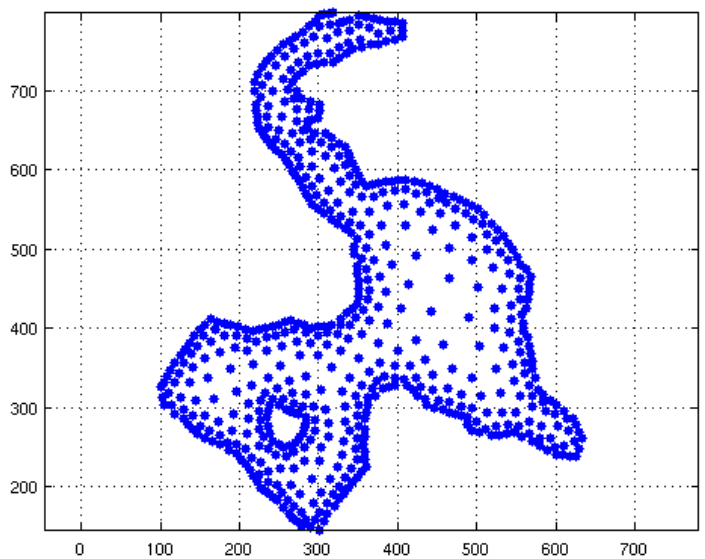
Here is an example of such a problem, in which a biological researcher wishes to study the diffusion of a pollutant in a lake. Some of our ideas from the unit square simply don't help!



PLANE: General Geometry Nodes

Could we start with a regular grid, and then keep just those points that are inside the lake region? Away from the boundaries, this might be almost OK, (but not really!) but it doesn't place points on the boundary, and it does a terribly irregular job of sampling the geometry near the boundary.

We must return to this issue, but for the moment, let's just assume we've picked sample points for the interior and boundary somehow, and move on to the second and third geometric tasks.

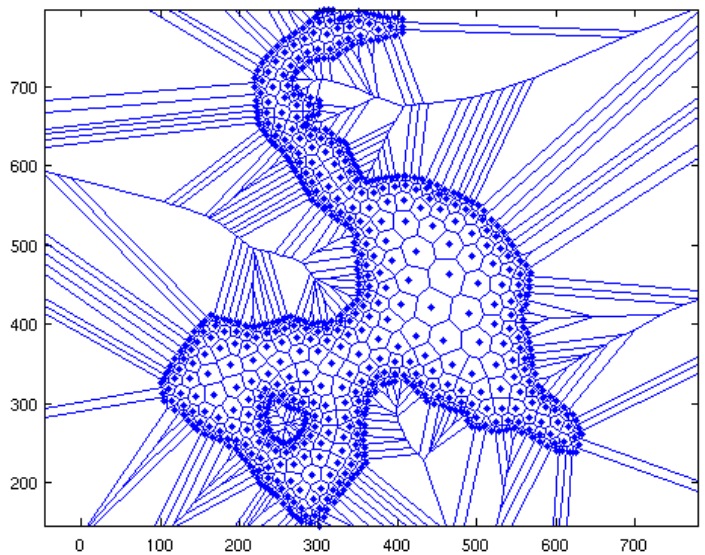


PLANE: General Geometry Elements

We need to break up the region into sets of points associated with each sample point, and the natural way to do this is to choose the nearest sample point.

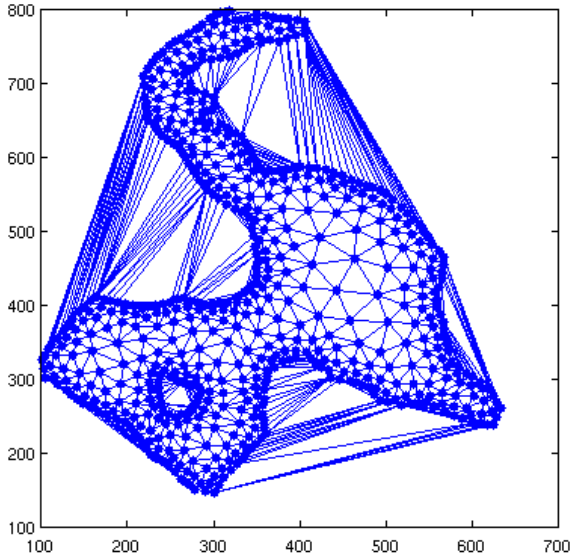
Such a subdivision by distance is known as a **Voronoi diagram**.

There are automatic procedures for computing the Voronoi diagram for a bounded, 2D region with polygonal boundaries
(...assuming the region lies in the plane!)



PLANE: General Geometry Triangulation

For our choice of points, we must also compute the triangulation, which connects triples of nearby points.



PLANE: Algorithm for General Geometry Elements

For a closed, bounded region $\mathcal{D}$ of the 2D plane, with the Euclidean distance function, with n distinct sample points p_i, we know that:

- to each g_i there corresponds a Voronoi region V_i;
- the union of the V_i is the region $\mathcal{D}$
- the interiors of distinct Voronoi regions have no intersection;
- for generator near the boundary, the boundary of V_i may include part of the boundary, but the remainder of the boundary is polygonal;
- away from the boundary, each V_i is a convex polygon;

PLANE: General Geometry Triangulation

We can say that two sample points p_i and p_j are "connected" if their corresponding regions V_i and V_j share a boundary of positive length. (Common boundaries of zero length are exceptional, but can be handled.)

To define the connections, then, we can simply draw lines from each point to its neighbor across their common boundary.

Such a procedure is easy to do by sight, but it may not be obvious that it can be computed, or how such information can be stored computationally.

In fact, given a set of points p_i in the plane, there is a standard geometric object known as the Delaunay triangulation, which records exactly this information.

PLANE: General Geometry Triangulation

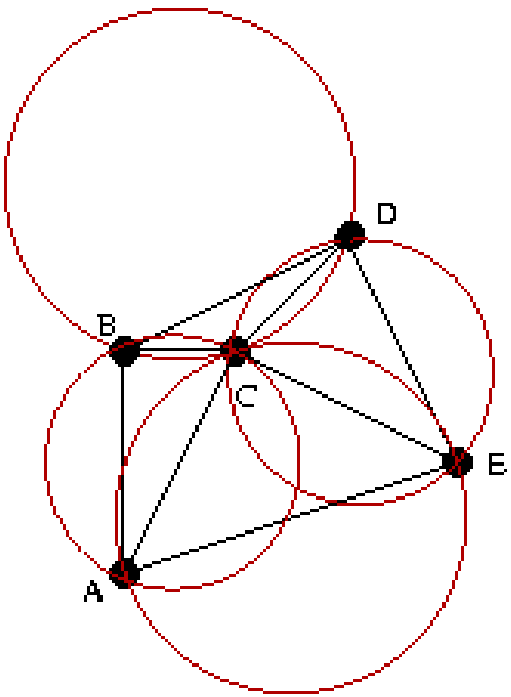
It is called a triangulation because the pattern of connections can be seen as a collection of groups of three nodes.

The Delaunay triangulation has the following properties:

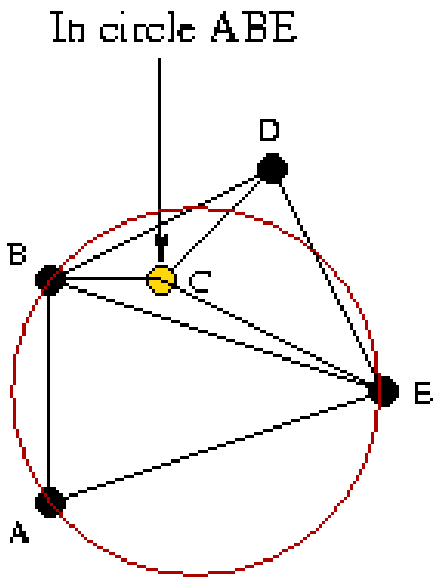
- the triangulation is maximal; no more triangles can be added to it without resulting in an illegal crossing of two connections;
- consider all maximal triangulations of the given points; for each triangulation, record the smallest angle over all its triangles; the Delaunay triangulation achieves the largest value for this measure.
- the triangulation can be described by triples of sample point indices;
- the Delaunay triangulation and the Voronoi diagram are dual graphs;
- if three sample points form one of the Delaunay triangles, then the circle through those points ("circumcircle") does not include any other points in its interior (we'll need this condition later!)

PLANE: General Geometry Triangulation

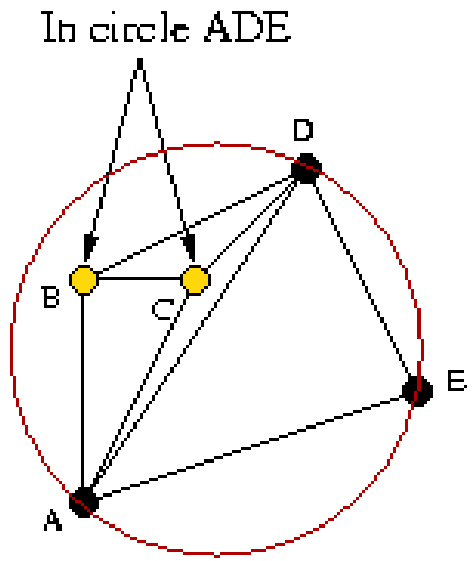
The empty circumcircle condition: a triangulation of a set of points has the Delaunay property if and only if the circumcircle for each triangle in the triangulation never properly contains a fourth point of the set.



Delaunay Triangulation



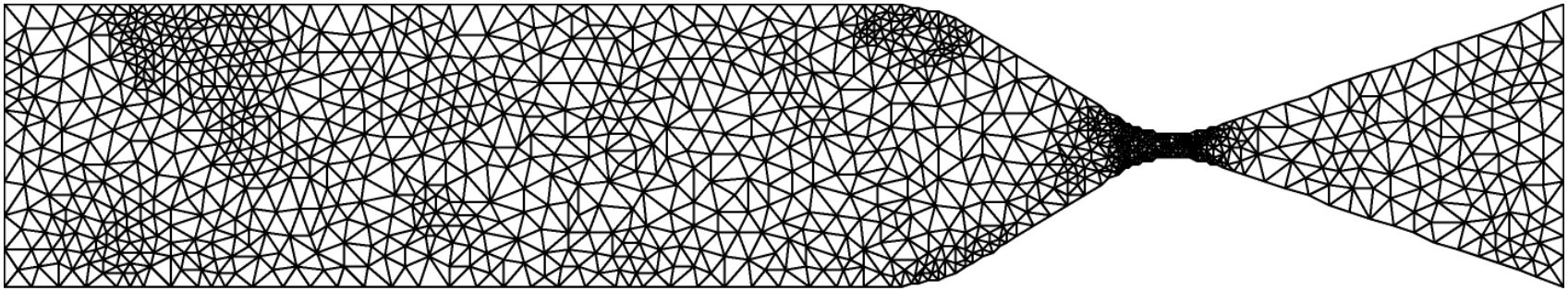
Non-Delaunay Triangulation



Non-Delaunay Triangulation

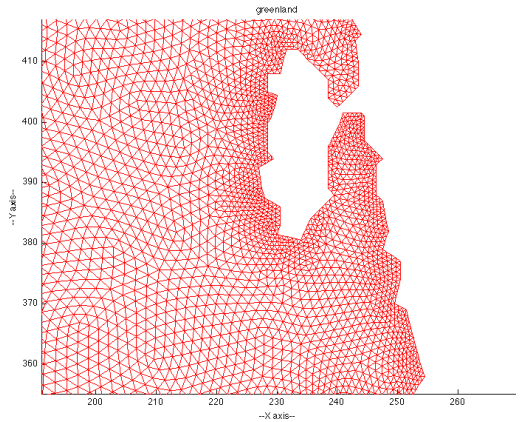
PLANE: General Geometry Triangulation

A fast and efficient program for computing the Delaunay triangulation of a set of points in the plane is available in the **triangle** library by Jonathan Shewchuk.



Source: Shewchuk (2005)

This library is so reliable and efficient that later, when we are working on a sphere, we will be searching for some way to redefine our problem so that **triangle** can help us.



PLANE: Dual Graphs

The Delaunay triangulation and the Voronoi diagram are dual graphs.

If we regard our element mesh as fundamental, then the dual mesh creates a node for each area, and connects nodes when the corresponding areas are in contact, creating a triangulation.

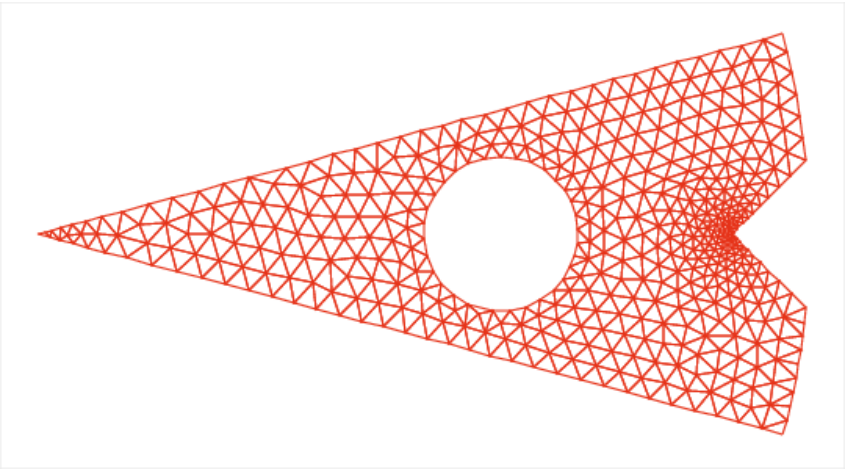
If we imagine the mesh as a decomposition of a country into territories, each with a capital city, then the dual triangulation may be thought of as the network of roads that connect capitals of adjacent territories.

Conversely, if we have a planar graph of capital cities and their connections, then the boundaries of the corresponding countries are found by drawing the perpendicular bisectors of the capital roads.

(This is easier to sketch on the board then to illustrate on a slide!)

PLANE: Well-Distributed Nodes

Good elements and triangulations depend on starting with well-distributed nodes. This is hard for peculiar geometries, regions with holes or small corridors. One procedure, by Strang and Persson, is a MATLAB program called **distmesh**.



The user defines the boundary of the region. The program places a regular array of points within the region, connects them with the Delaunay triangulation, and treats these connections like repulsive springs. Points near the boundary are missing a neighbor on that side and move closer. The mesh naturally adjusts to the geometry in a beautiful way.

PLANE: Well-Distributed Nodes

We will prefer a related method that is more closely tied to the Voronoi diagram.

We start with an arbitrary scattering of nodes in the region. We construct the Voronoi diagram. We replace the original nodes by the centroids of the Voronoi polygons. This is like moving the capital city to the center of the territory. But when we move the capital cities, we must recompute the Voronoi diagram. Repeating this process long enough, the nodes settle down to a well-distributed pattern known as a Centroidal Voronoi Tessellation or **CVT**.

Let us watch this process in an animation, in which we start with the worst possible initial configuration, namely, all the points crammed into one tiny region.

Animation: **cvt_movie_p08_cramped.mov**

PLANE: CVT Algorithm #1

The basic CVT algorithm in 2D is easy to describe:

```
Choose n points p at random in the region;
while ( not satisfied )
  compute the Voronoi diagram for the points p;
  for each p(i), compute c(i) = Voronoi polygon centroid;
  replace all p by c.
```

The results can be beautiful, and it can be shown that they minimize an energy function. Improvements occur quickly at the beginning, but then slow down. A more serious issue is that, depending on the circumstances, the algorithm may not be easy to implement in practice:

- Voronoi algorithms are not always available, convenient, or efficient. (This becomes a real issue in higher dimensions.)
- Unless the boundary is polygonal, some Voronoi polygons will actually have curved edges.
- Weighted or nonuniform meshes cannot be generated this way.

PLANE: What is a Nonuniform Mesh?

Surely mesh uniformity is a good thing - why would we ever want a nonuniform mesh?

We might like to create nonuniform meshes in order to have more points near the boundary, or a transition range where sudden changed are expected. For an advection problem, we might to grade the mesh to correspond with the flow gradients.

Typically, a user requests a nonuniform mesh by specifying a mesh density function $h(x, y)$ over the region, that indicates the relative density of the mesh at each point. Taking $h(x, y) = 1$ as a base value, then higher and lower values in a region will request more or fewer mesh points.

In the simplest case, we can restrict the mesh density to affecting only the location of the nodes, and compute the elements and triangulation in the usual way. A version of the CVT algorithm can easily handle this initial requirement.

PLANE: CVT Algorithm #2

This CVT algorithm uses sampling, and hence only approximates the correct result. The user supplies a procedure **sample()** returning points in the region distributed according to the desired density.

```
p = sample ( n );    <-- Starting points
while ( not satisfied )
  s = sample ( big number );  <-- Approximate the geometry
  c(i) = average of all points nearest to p(i);
  replace all p by c.
```

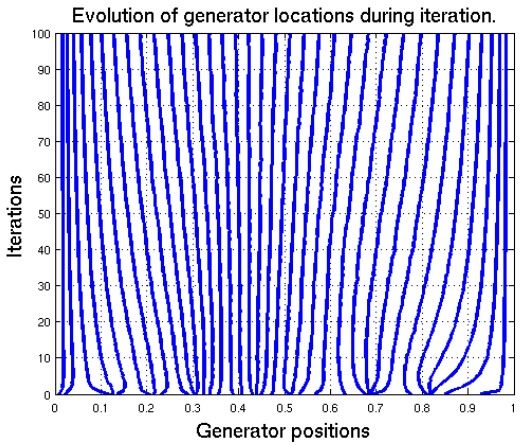
We are essentially replacing the Voronoi diagram and centroid calculation by a Monte Carlo estimate. The mesh density function shows up in the integral implicitly, through the user's **sample()** function.

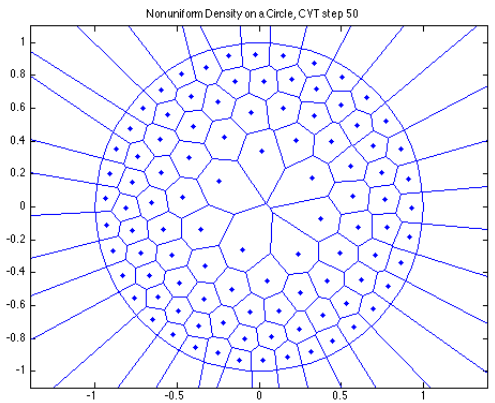
This procedure may be approximate, and somewhat slow but:

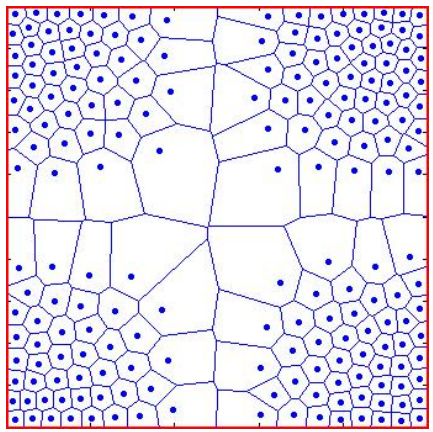
- it can be carried out in any geometry that the user can sample;
- it can impose any mesh density that the user can formulate;
- it works the same way in any spatial dimension;
- it can work on any surface or constrained region for which a distance can be consistently defined;
- there is no need for external Voronoi or triangulation software.

PLANE: Nonuniform Meshing

The first plot shows the evolution in time of a 1D CVT with nonuniform density. The second and third plots are converged results for nonuniform densities in the circle and square.







PLANE: The 2D Planar Problem is NOT a Problem

With the CVT algorithm to provide nodes that are distributed uniformly, or according to a nonuniform pattern we specify, we can reliably compute our elements and connections, and hence set up a mesh on a region in a general 2D geometry;

Because the CVT algorithm extends to other dimensions, we can use similar procedures in 3D, for instance.

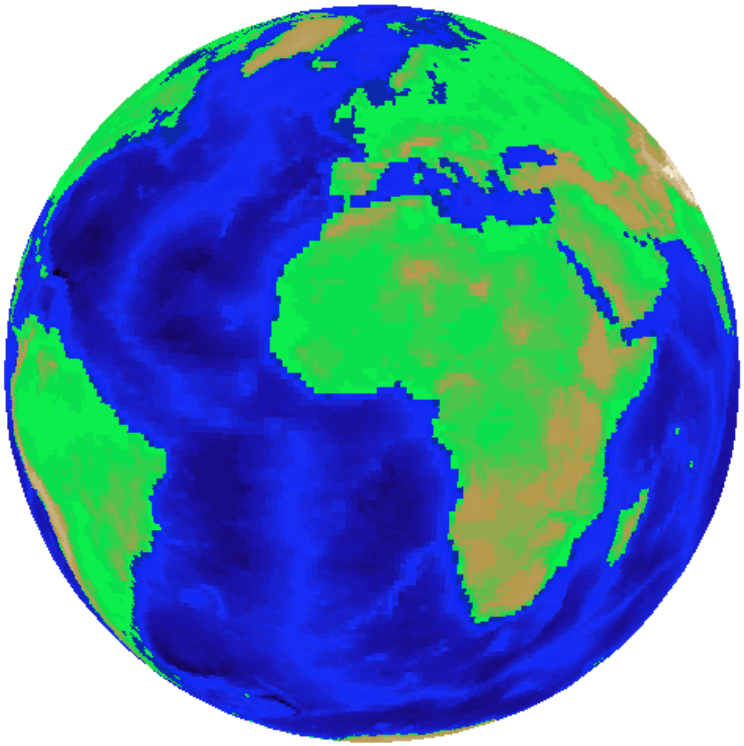
But now we will consider trying to transfer our procedures to a region so familiar that we often mistakenly think of it as a plane, that is, the 2D curved spherical surface we call the earth, on which many of our algorithms will simply **collapse**.

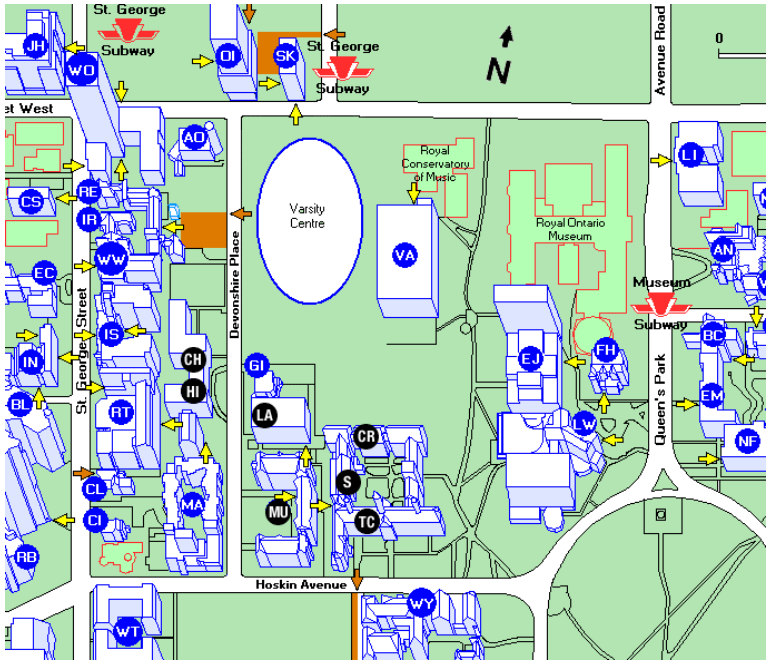
The Spherical Mesh

- Planar Meshes
- **The STRIPACK Sphere Mesh**
- The SCVT Sphere Mesh
- Examples

STRIPACK: The Uncooperative Surface

We live on a sphere...





...but the sphere is so large that locally, it seems **flat**.

STRIPACK: The Uncooperative Surface

Among the peculiar properties of a sphere, we note that

- "straight" lines on a sphere are great circles;
- therefore, all straight lines must intersect;
- there is no point at infinity, polygons don't really have "insides";
- the sphere cannot generally be covered by congruent triangles or polygons;
- two triangles on a sphere are similar (same angles) only if they are congruent (same size)!
- the average of points on the sphere is not itself on the sphere;
- circles around a fixed point grow with radius, then get *smaller*;
- a circle of latitude does not represent a line of shortest distance;
- spherical triangle angles can sum to anything between π and 3π;

STRIPACK: Well-Distributed Points?

As one indication of the difficulty of working on a sphere, we cannot, in general, answer the question of how to arrange n points on the sphere so that they are maximally separated. This is an open research question, and people have tabulated their best results, so far, for arrangements.

- The Tammes problem asks for the maximal size of n nonverlapping circles on the sphere.
- The Thomson problem asks for the arrangement of n electrons on the surface of the sphere which minimize the total Coulomb energy.

Only a few best results have actually been proven. Cases up to a few hundred points have been studied.

STRIPACK: We Need Millions of Nodes

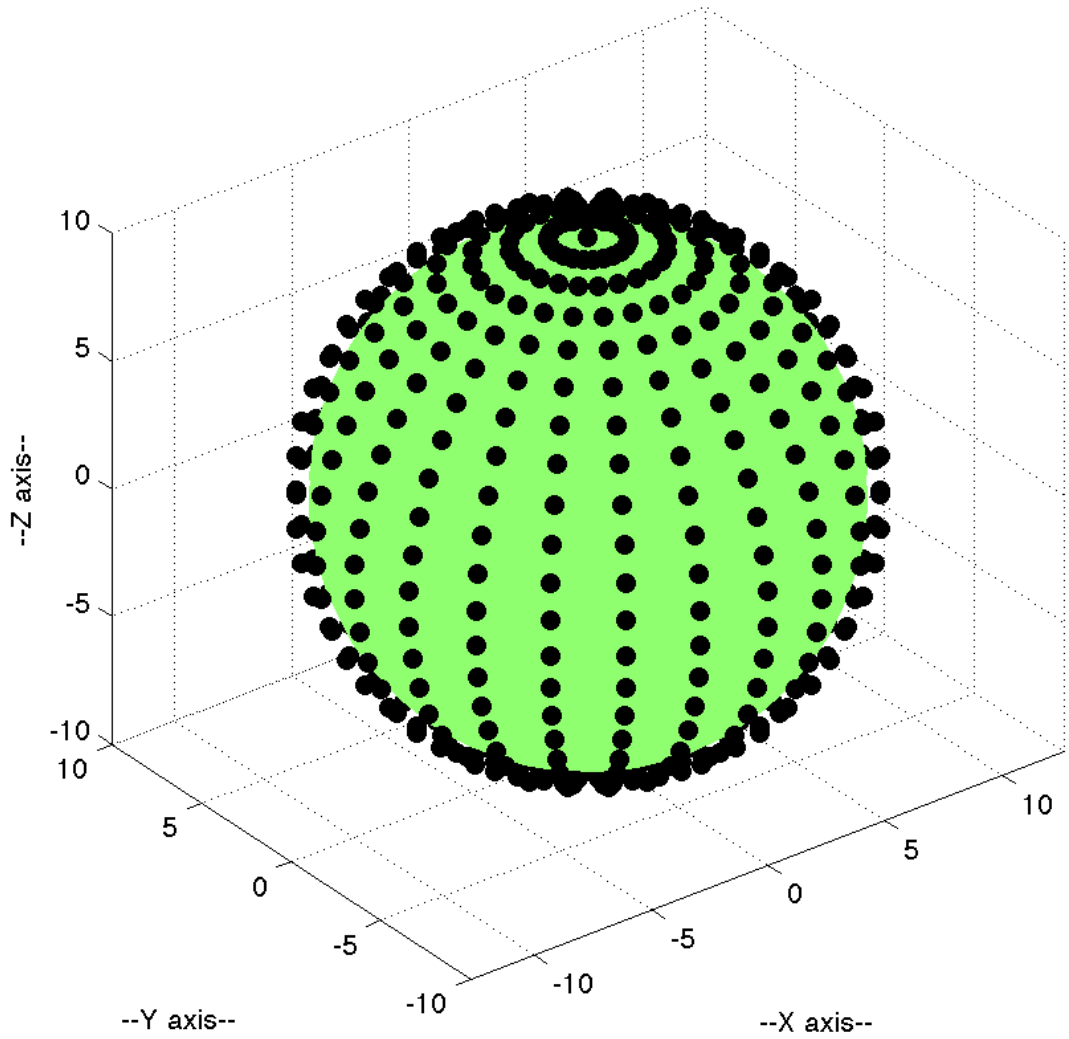
The sphere is a difficult surface to work on, and we need to do **a lot** of work there. We are working with a weather and climate modeling group at Los Alamos, whose **MPAS** software simulates the partial differential equations for the atmosphere over the entire globe, and for this they need a good mesh.

Their current standard requires meshes whose elements are about 15 kilometers $\approx$ 10 miles on a side, or roughly 200 square kilometers in size. The surface area of the earth is about 510 million square kilometers, which means we would need about 2 million elements, defined by nodes for which we can confidently say that they are about 15 kilometers apart.

Unlike the plane, good meshes are hard to create on the sphere!

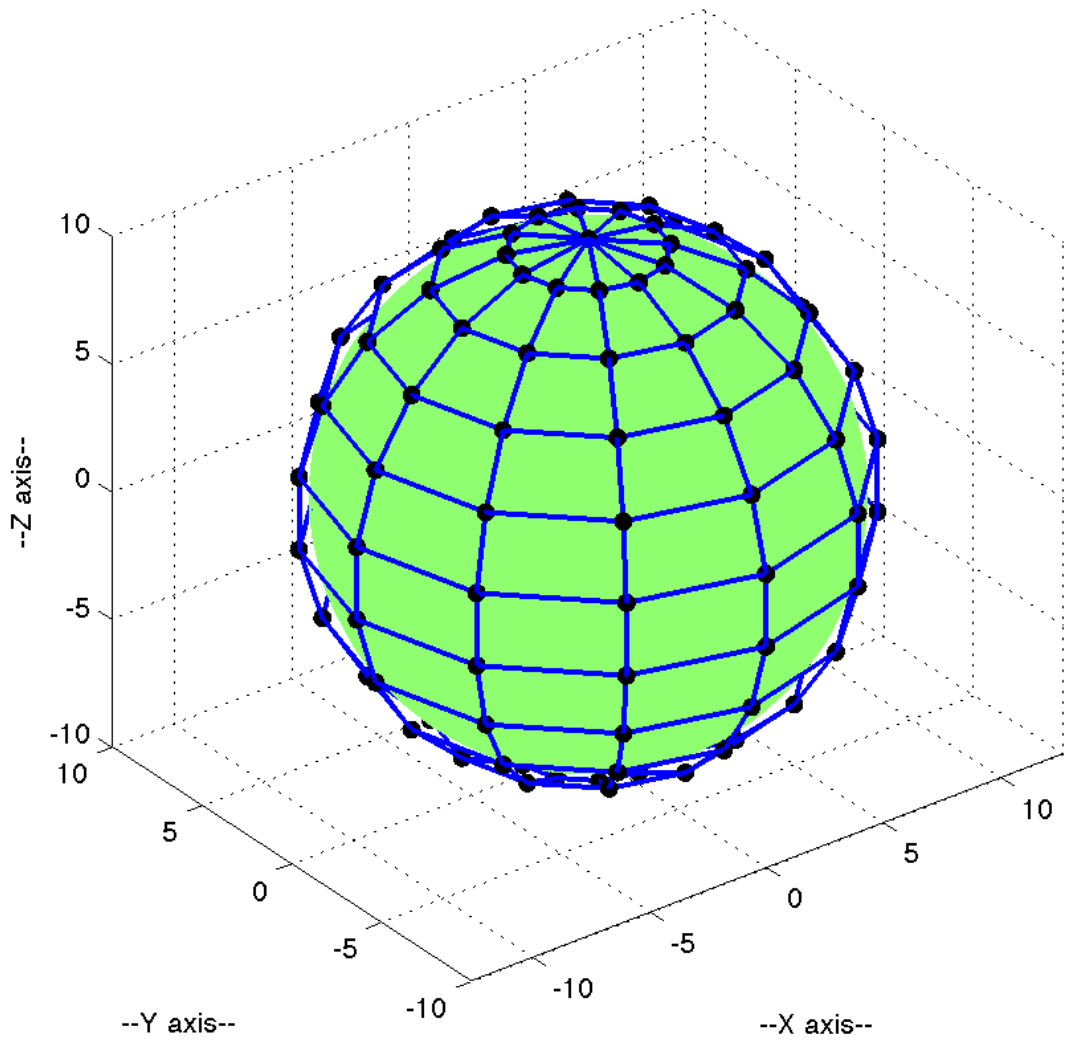
STRIPACK: The Latitude/Longitude Mesh

Perhaps the spherical mesh we are most familiar with is the latitude/longitude grid.



STRIPACK: The Latitude/Longitude Mesh

Connecting lines of latitude and longitude gives us a quadrilateral grid, except at the poles.



STRIPACK: The Latitude/Longitude Mesh

The LL grid has several advantages:

- the grid is easy to generate, for any level of refinement;
- the elements are logical squares, except at the poles;
- the LL grid provides a coordinate system;

However...

- Latitude and longitude lines seem to be orthogonal...but they aren't!
- The grid includes preferred directions;
- The LL coordinate system becomes singular at the poles;
- The elements vary **enormously** in size and shape;
- As we refine the mesh, elements touching the poles include a tiny angle;
- Having two element types (squares and triangles) is a minor annoyance.
- As a product grid, local refinements are not easy.

STRIPACK: The CVT Option

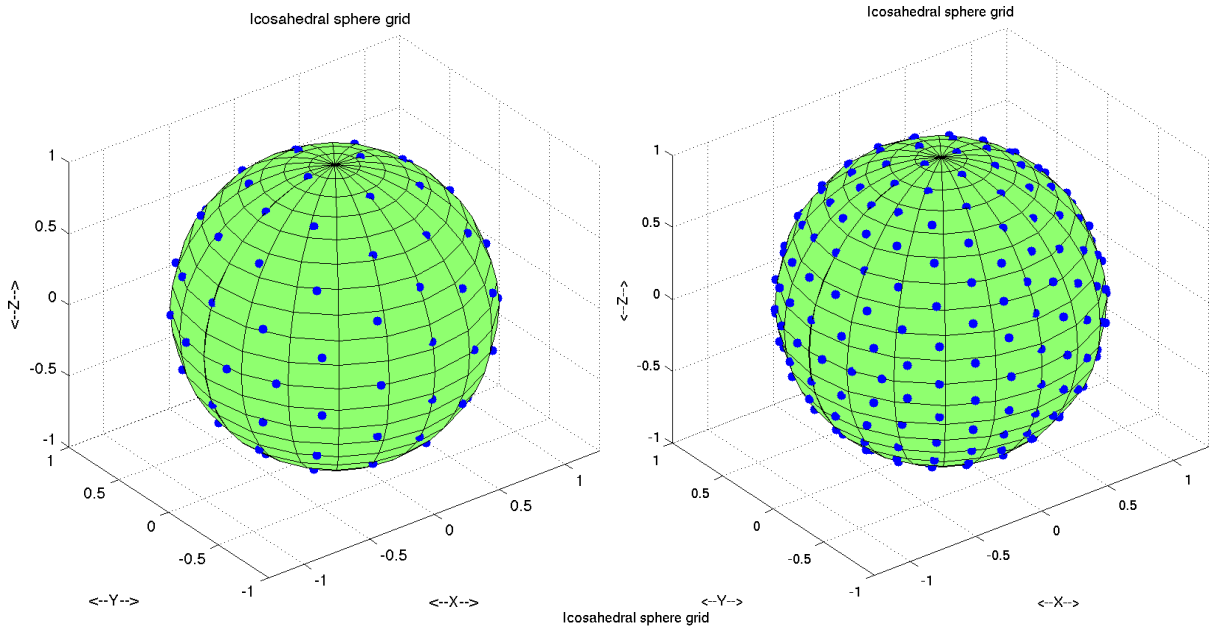
There are many ways of trying to set up a grid on the sphere. Our experiences with the 2D plane suggest that the CVT approach was very flexible there.

But our concerns are:

- Can we determine a good initial grid, so that the CVT iteration converges rapidly?
- Do we have the mathematical analysis and software tools necessary to carry out a version of the CVT algorithm on the sphere?
- If so, can the CVT algorithm produce a satisfactory mesh of millions of points in a reasonable amount of computing time?

STRIPACK: Initial Grid

The 12 vertices of the icosahedron are perfectly separated on the sphere. If we triangulate these vertices, we get 20 faces. If we bisect each edge, we can replace each face with four smaller ones, which are no longer congruent, and no longer "perfectly" placed. As we repeatedly refine this grid by bisection, the mesh degrades, but is still very acceptable as a starting point.

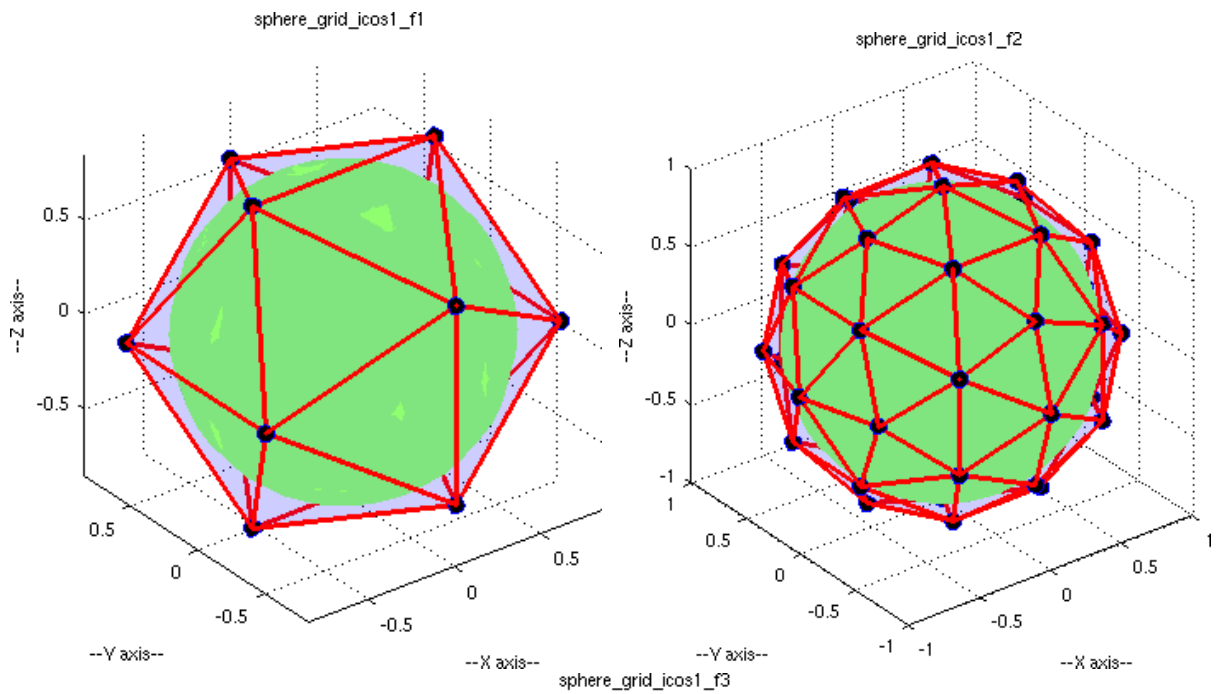


STRIPACK: The STRIPACK Program

stripack is a Fortran77 program, written by Robert Renka, and available as ACM TOMS algorithm #772.

Given a set of points on the unit sphere, **stripack** can compute the vertices of the Voronoi polygons around each point, or it can organize the points into Delaunay triangles.

Thus, **stripack** enables us to carry out the CVT algorithm for nodes, and then to organize the sphere surface into elements and connections.



STRIPACK: CVT Initialization

To achieve a 15 kilometer element width on the Earth's surface, we need about 2,000,000 elements of uniform size. The starting nodes for the CVT iteration are created by "bisecting" an icosahedral set of nodes 9 times.

Step	Nodes
0	12
1	42
2	162
3	642
4	2,562
5	10,242
6	40,962
7	163,842
8	655,362
9	2,631,442

STRIPACK: Modified CVT Algorithm

We outline how the grid initialization and **stripack** can fit into a CVT algorithm for the sphere:

```
Choose n initial points p using the bisection grid;

while ( not satisfied )
  stripack computes the Delaunay triangulation for points p;
  implicitly compute the Voronoi diagram for points p;
  for each p(i), compute c(i) = Voronoi polygon centroid;
  replace all p by c.
```

STRIPACK: STRIPACK Deficiencies

For our multi-million node spherical mesh problem, however, **stripack** has some serious deficiencies:

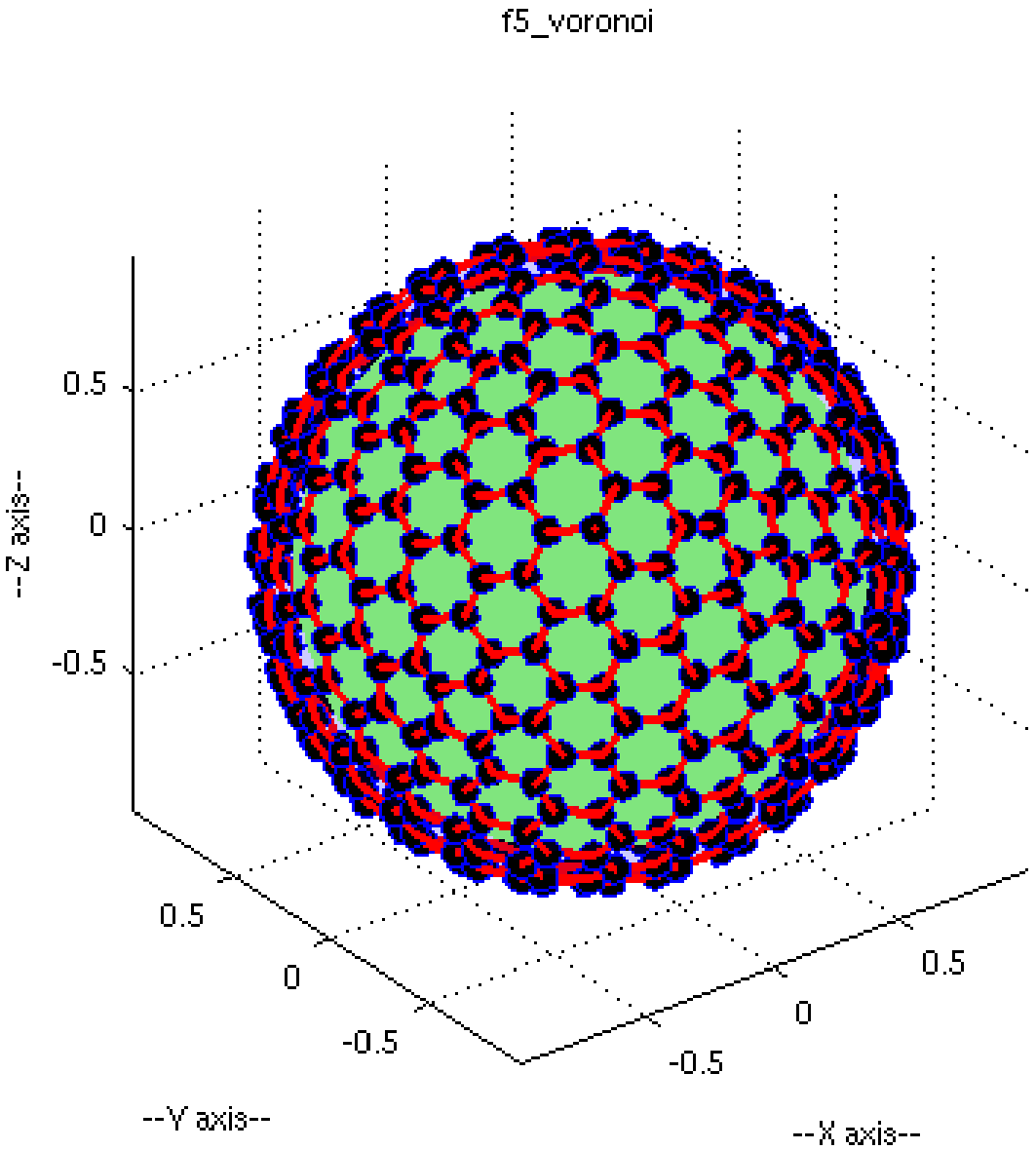
- it is written in a specific, outdated programming language;
- the algorithm is slow;
- it is not amenable to parallel programming;
- it does not include the ability to deal with subregions of the sphere,
- it cannot handle variable mesh densities.

For our problems, a CVT mesh requires hundreds or thousands of iterations. Each iteration requires the computation of a Delaunay triangulation, that is, a call to **stripack**. Time efficiency is a serious issue, so we benchmark our **stripack** algorithm by timing it over a single triangulation. In a production run, the problem will be larger, and will be triangulated thousands of times!

It takes about 208 seconds (≈3.5 minutes) for **stripack** to triangulate the level 7 mesh with 163,842 nodes, on an Intel Core 2 CPU.

STRIPACK: The STRIPACK Program

So **stripack** can do what we need, but it just takes a long time!



The Spherical Mesh

- Planar Meshes
- The STRIPACK Sphere Mesh
- **The SCVT Sphere Mesh**
- Examples

SCVT: No Speedup Without Parallelism

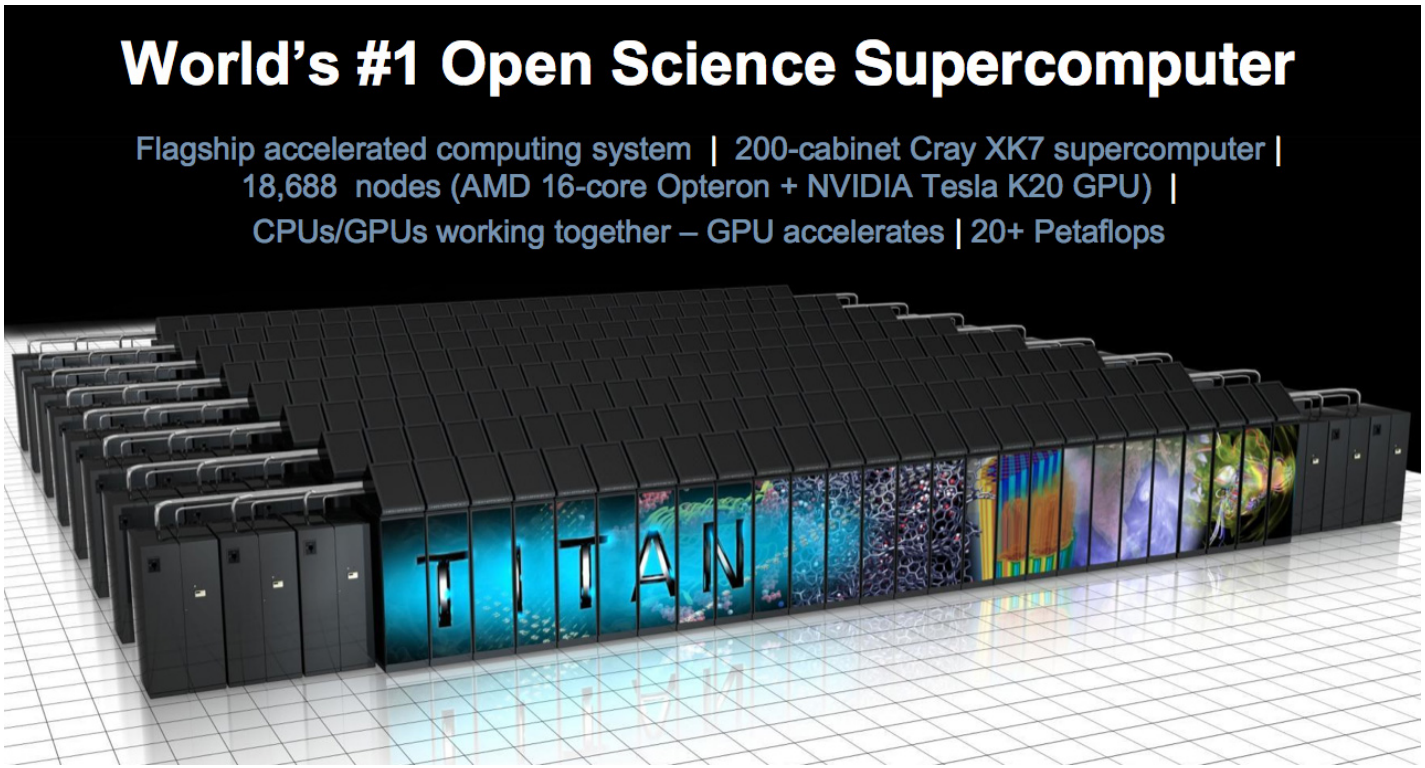
Programs for enormous problems like weather prediction must run extremely fast. Their results are so valuable that huge data centers are constructed for them. The weather prediction program has expanded to run over a model of the entire Earth; now it is repeatedly shrinking the size of the elements used in the model.

For decades, the program's appetite for computing power could be matched by the increase in computer processor speed and power. However, an absolute performance ceiling was reached sometime around the year 2005, when processor clock speeds of about 4 Gigahertz were achieved. **No processor will ever run significantly faster than this!**

The only way to keep computational science advancing requires parallel programming.

SCVT: No Speedup Without Parallelism

Modern "supercomputers" are now simply stacks of thousands of ordinary, stripped down personal computers, with extremely fast interconnections, which must solve parts of a common problem whose solution is then stitched together at the end.



SCVT: A Plan for Parallel Meshing

We considered trying to make a parallel version of **stripack**, but felt that the programming obstacles were severe.

On the other hand, parallel triangulation problem in the plane looked like a distinct possibility.

Suppose we could divide a planar region into subregions, in such a way that every node fell into at least one subregion. Separate processors could compute the triangulation of each subregion independently - perhaps invoking the **triangle** program to do this.

So if we had 10 processors, we might do this task 10 times faster.

Then we could just patch the individual triangulations together.

SCVT: Independent Triangulation of Subregions

But the triangulation pieces would disagree along their boundaries, because each subregion would only be able to work with the nodes it contained.

Recall that a Delaunay triangle is correct when its circumcircle contains no other nodes.

So, for our patchwork triangulation, we are only sure of the triangles that are some distance away from the interfaces between regions.

But we can work around this problem. We expand the subregions so that they have an extensive overlap. We compute the triangulation in a subregion. But then we toss away all triangles whose circumcircle extends outside the subregion. The remaining triangles are guaranteed to be correct; the overlap will only cause us to compute some correct triangles more than once.

SCVT: Sharing the Updated Node Positions

Another issue can arise if we are carrying out the CVT algorithm. Each step of the algorithm starts with a set of nodes p, and replaces them by the centroids c of their Voronoi polygons.

If we are working in parallel, then the processors need to agree on where all the nodes are. Each node "belongs" to just one processor, but, if it is in an overlap region, its position must be available to other processors. Once the CVT update step is completed, each processor informs its immediate neighbors of the updated node positions.

In some cases, a node will move so much that its "ownership" will actually be transferred from one processor to another.

SCVT: Voronoi Diagrams from Delaunay Triangulations

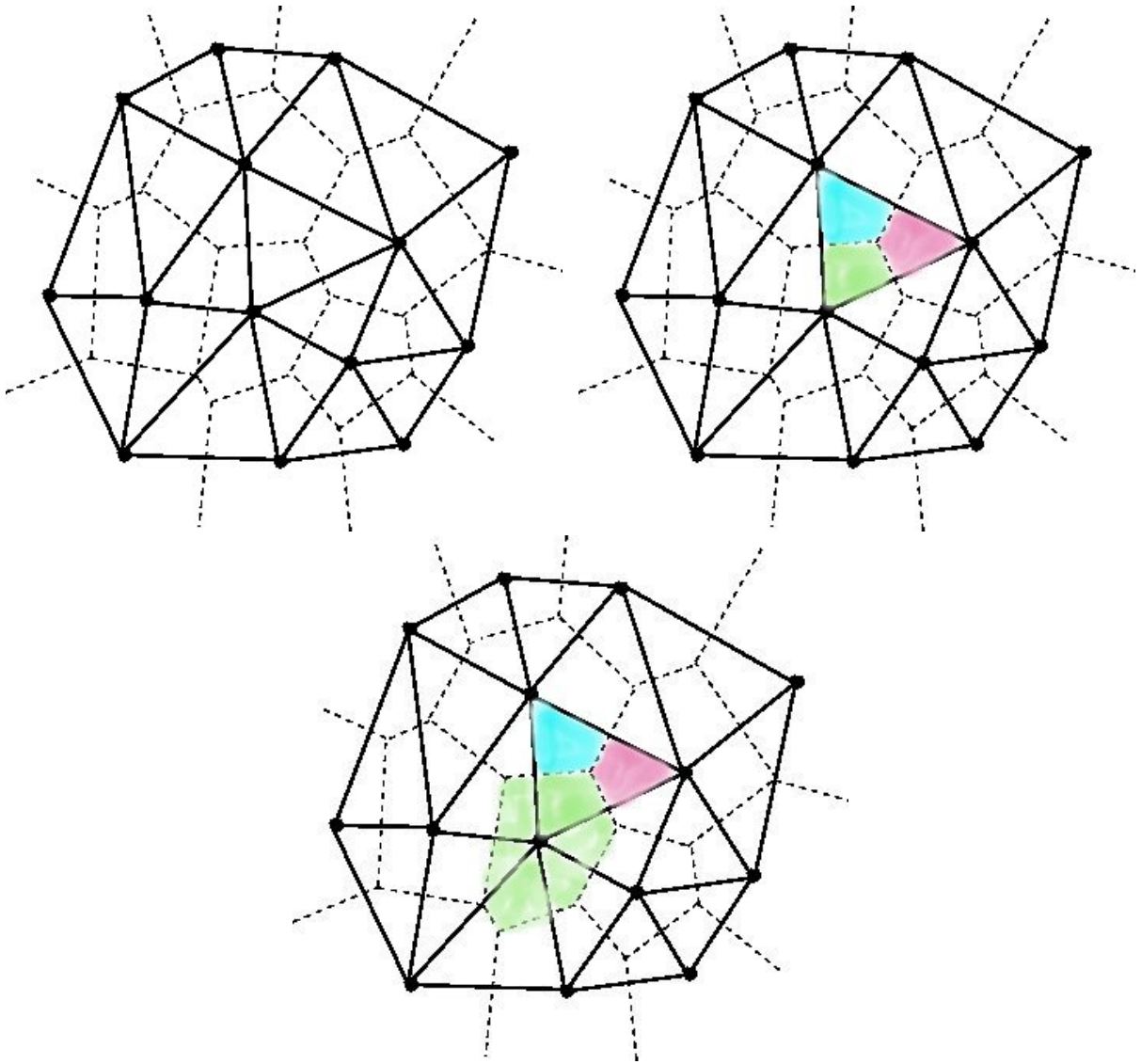
While **stripack()** can compute both Voronoi diagrams and Delaunay triangulations, we are going to rely on the **triangle()** library, and just get a triangulation back.

But the CVT iteration requires us to construct the polygons associated with the Voronoi diagram, so that we can compute their areas and centroids.

Since the two objects are dual graphs, it should not be surprising that, if we are careful, the triangulation can be mined for the data we need.

SCVT: Voronoi Diagrams from Delaunay Triangulations

Every triangle splits at its circumcenter into three pieces (red, green, blue). Each piece forms part of a Voronoi polygon (all the green pieces).



SCVT: From Sphere to Plane

What we have said, so far, means that, if we plan carefully, a parallel program can compute triangulations, Voronoi diagrams, and hence a CVT, using the efficient **triangle** program...*as long as we are working in the plane.*

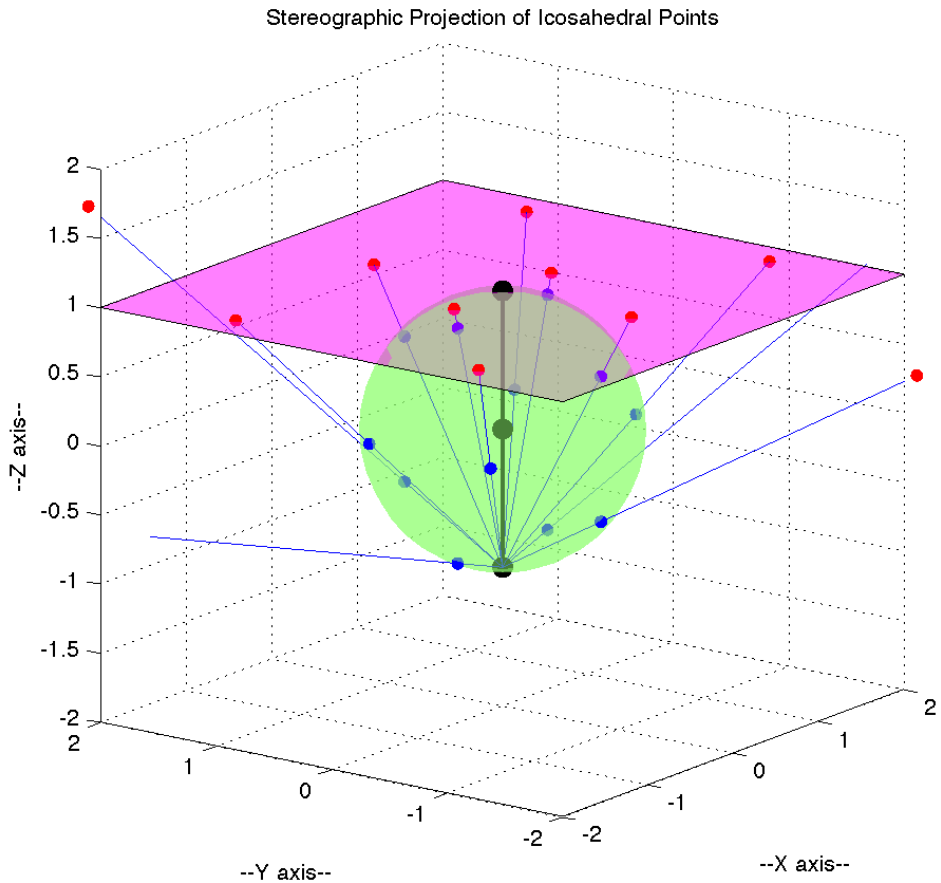
But, of course, our climate modeling problem is set on the sphere, not the plane. Luckily, there is a beautiful, simple conformal mapping technique that can translate back and forth between problems on the sphere and problems in the plane, called stereographic projection.

Such a mapping will necessarily distort many properties of the point sets it operates on. However, as we will see, it happens to preserve exactly the one property that we absolutely need in order for this translation to be useful.

SCVT: Stereographic Projection

In stereographic projection, a plane is attached to the sphere at some point (which we can think of as the "north pole"). Points s on the sphere are mapped to points p on the plane by drawing the line from the "south pole" through s, and locating the intersection of this line with the plane.

Points near the south pole singularity are heavily stretched.

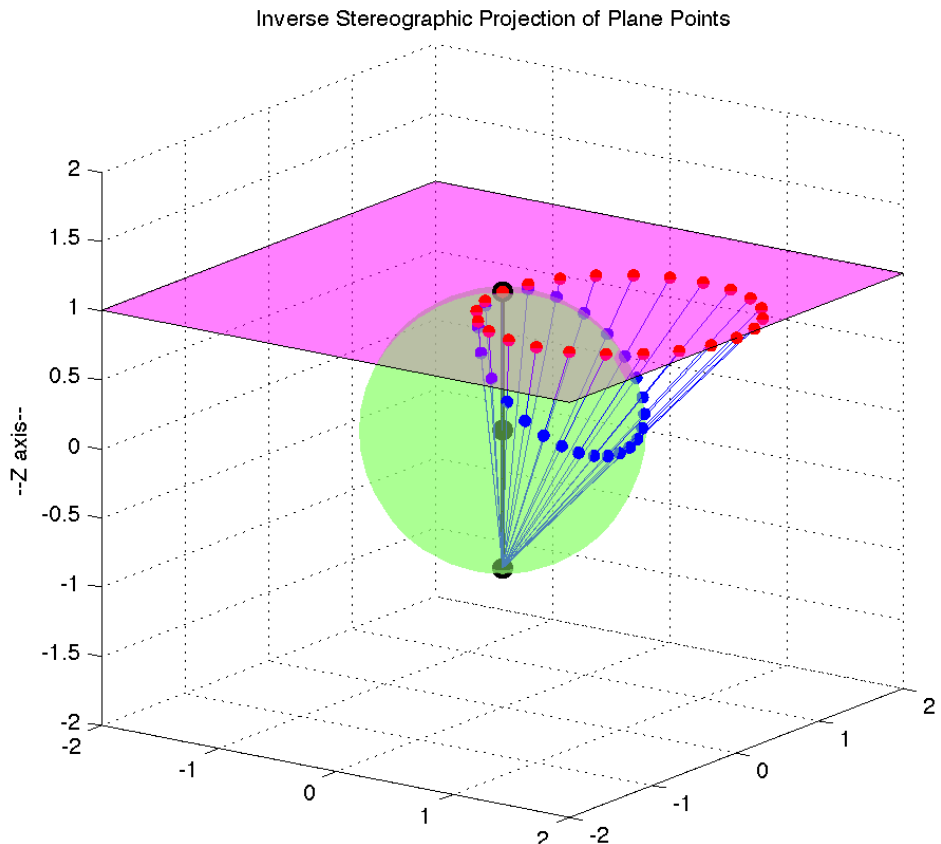


SCVT: Stereographic Projection

Circles on the sphere are mapped to circles in the plane and vice versa.

Delaunay triangulation is defined by the empty circumcircle condition, so stereographic projection preserves the Delaunay triangulation property.

We can start with data on the sphere, map it to the plane, triangulate it there, and employ the triangle information back on the sphere.



SCVT: The MPI-SCVT Program

We were able to convince the MPAS group that the SCVT ideas could be used to develop an alternative approach to computing meshes for spheres or portions of a spherical surface.

For his PhD thesis, Doug Jacobsen thought up, implemented, and tested many of these ideas in a program called **mpi-scvt**.

The code splits up the surface into separate regions, assigning one to each processor. Each processor knows its six nearest neighbors, with which it may have to communicate after each step of the CVT iteration. Only at the very end of the CVT iteration is it necessary for all the data to be gathered together into a single processor.

SCVT: The MPI-SCVT Program

He computed big meshes; meshes that included boundaries between land and ocean, and sequences of meshes for time dependent problems which had to focus on a moving disturbance.

He was able to match tabulated data, to model boundaries better, and to incorporate mesh density functions to grade the mesh.

His parallel code, based on **triangle()**, could run as much as 40 to 4,000 times faster than a sequential code based on **stripack()**.

The program is available at
https://sourceforge.net/projects/mpi-scvt/

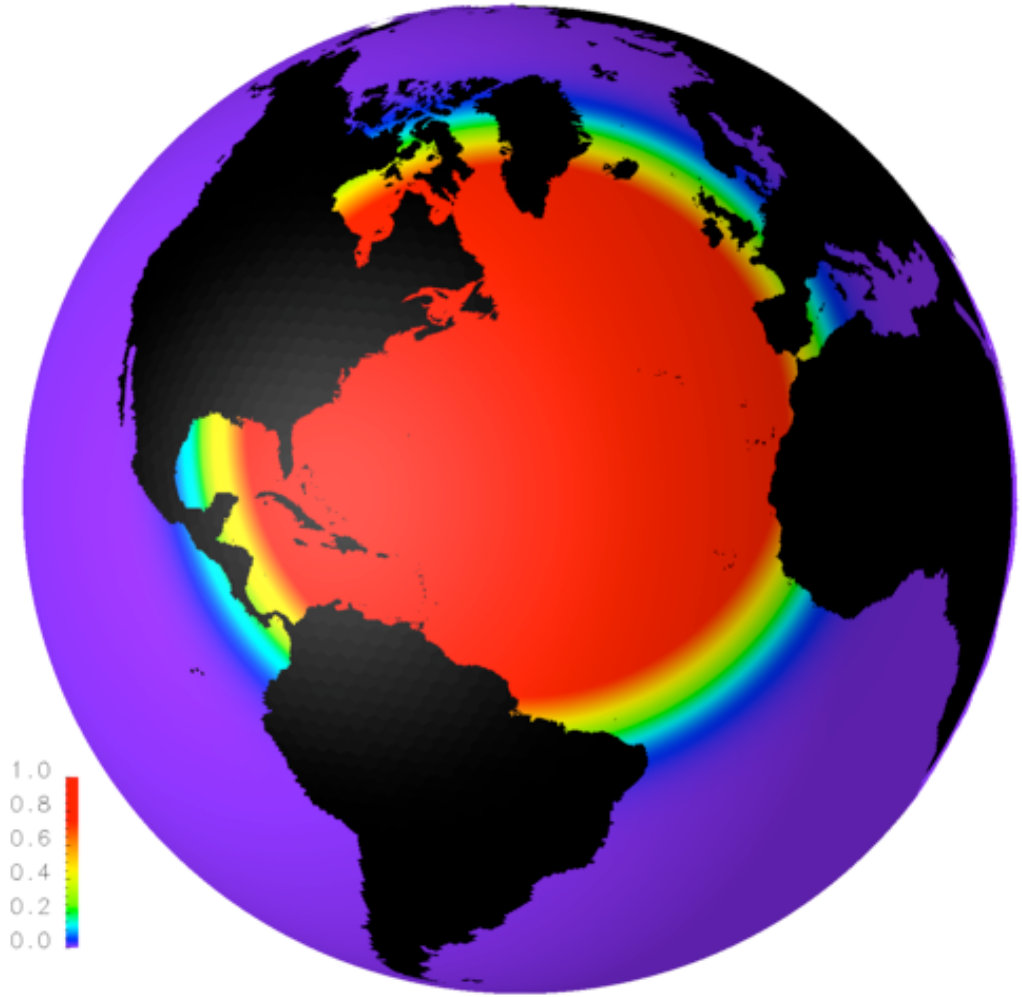
His thesis is available at
http://people.sc.fsu.edu/~jburkardt/pdf/jacobsen_thesis.pdf

Doug himself is now "available" at Los Alamos National Laboratory, working with the MPAS group.

The Spherical Mesh

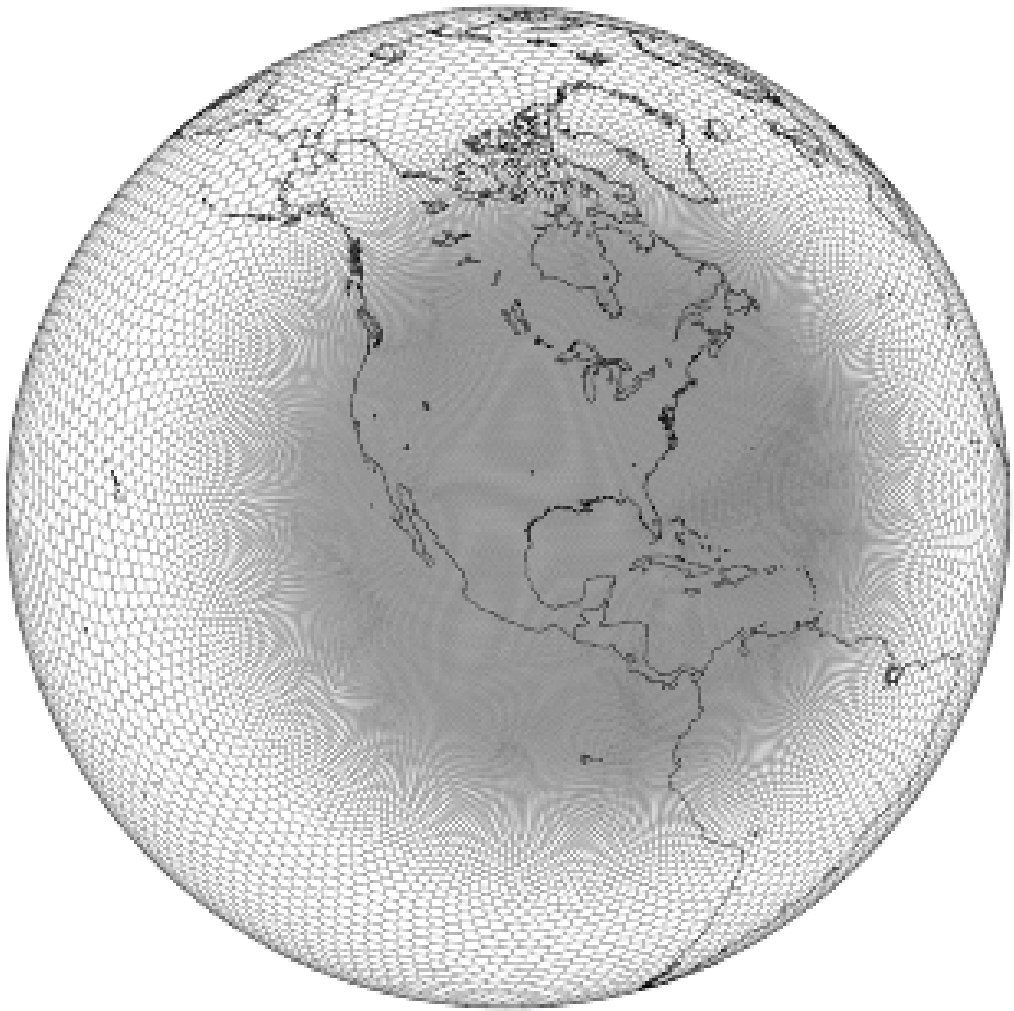
- Planar Meshes
- The STRIPACK Sphere Mesh
- The SCVT Sphere Mesh
- **Examples**

EXAMPLE: Atmosphere Density Function



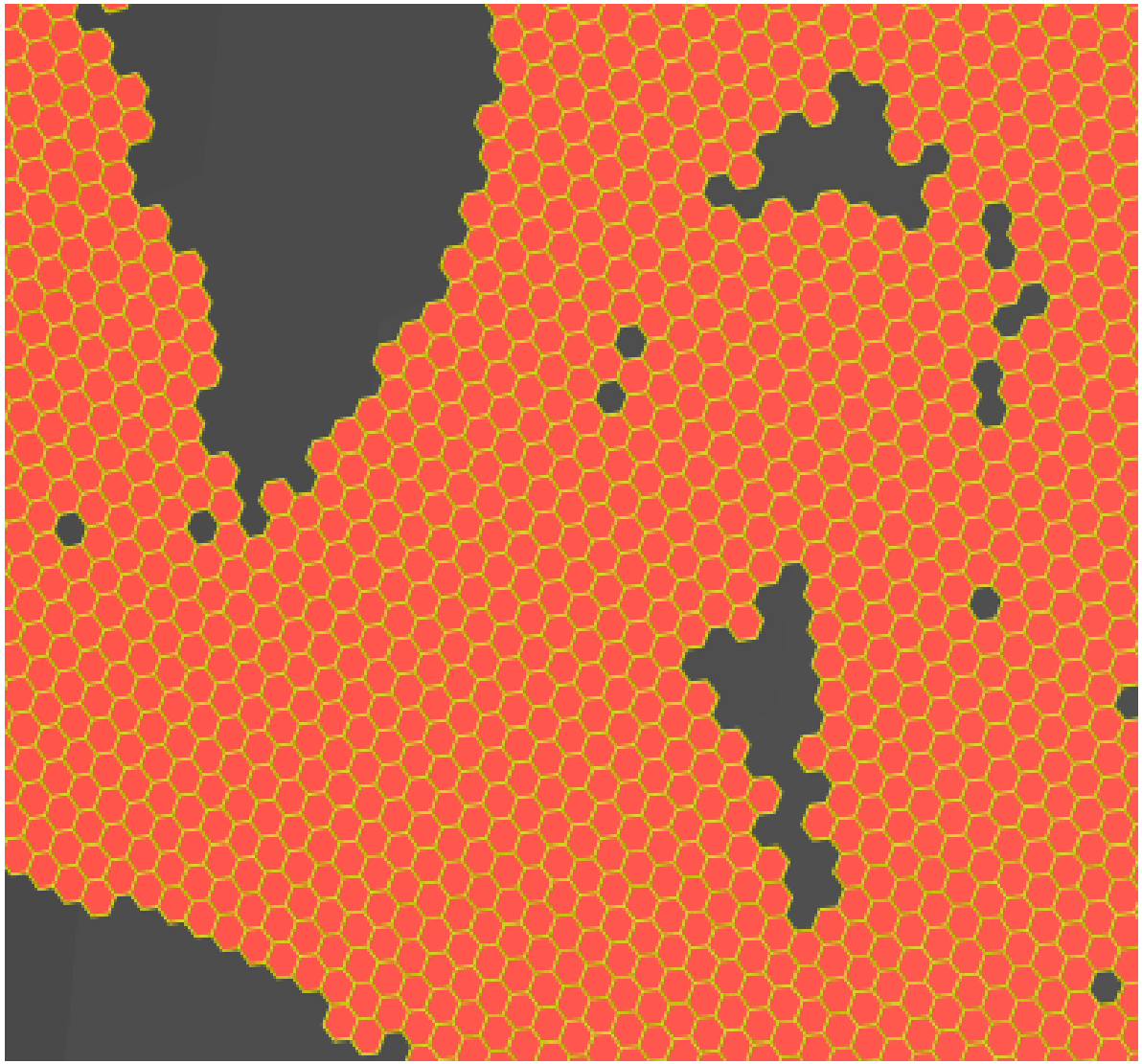
A density function for North Atlantic regional modeling

EXAMPLE: Atmosphere Grid



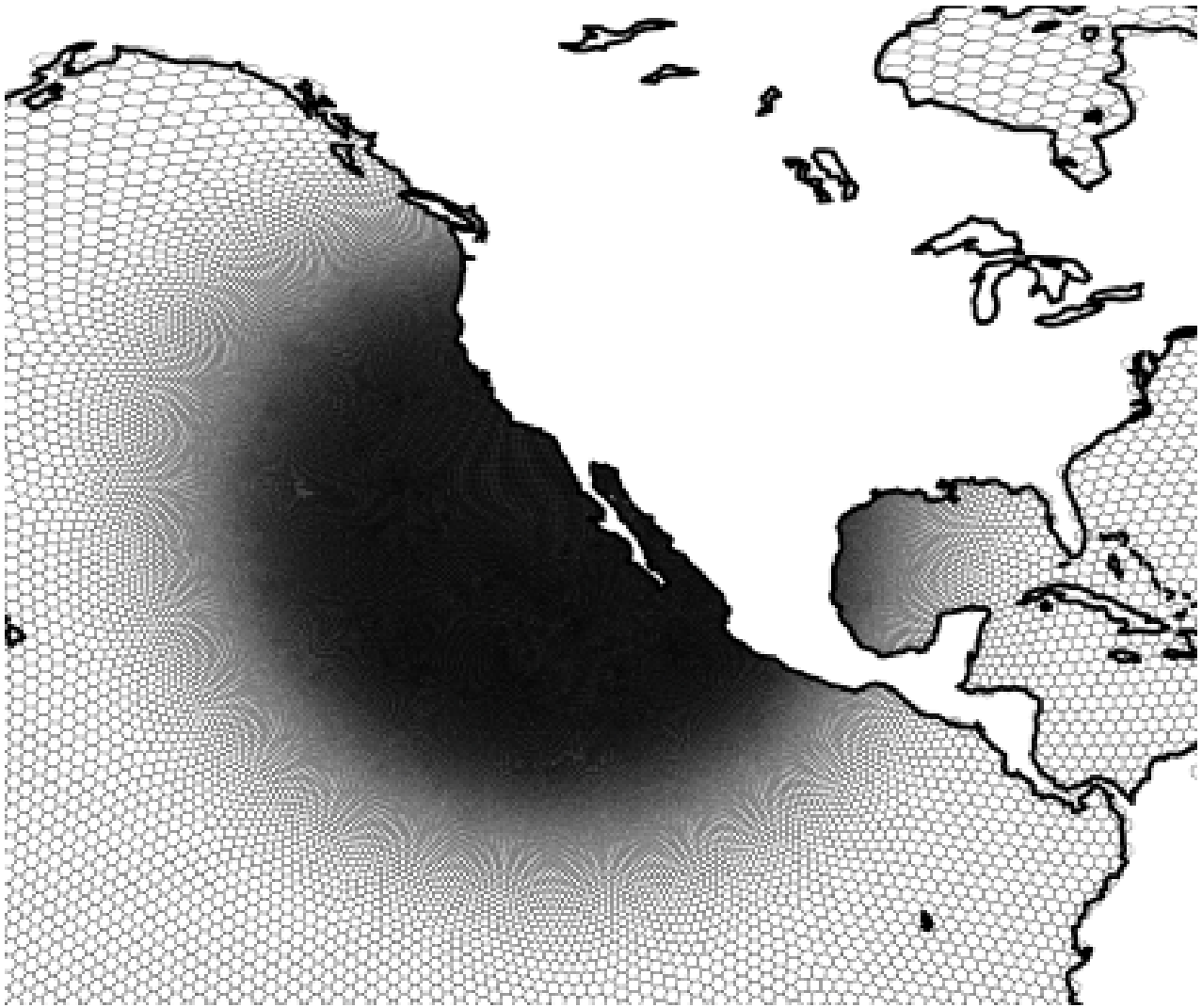
A grid for the atmosphere, with increased density over North America.

EXAMPLE: Florida Ocean Grid



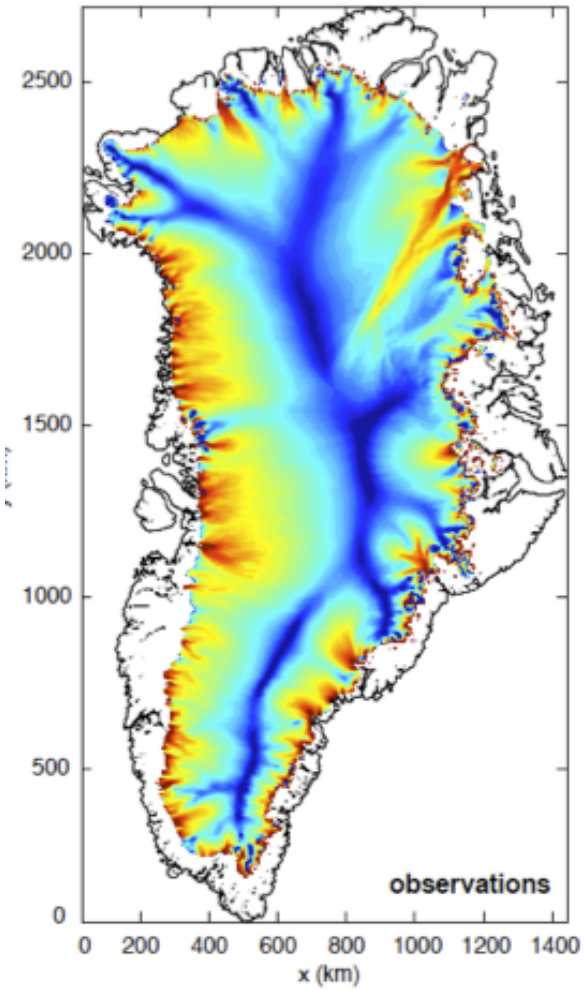
A closeup of an ocean mesh, with uniform mesh density, near Florida.

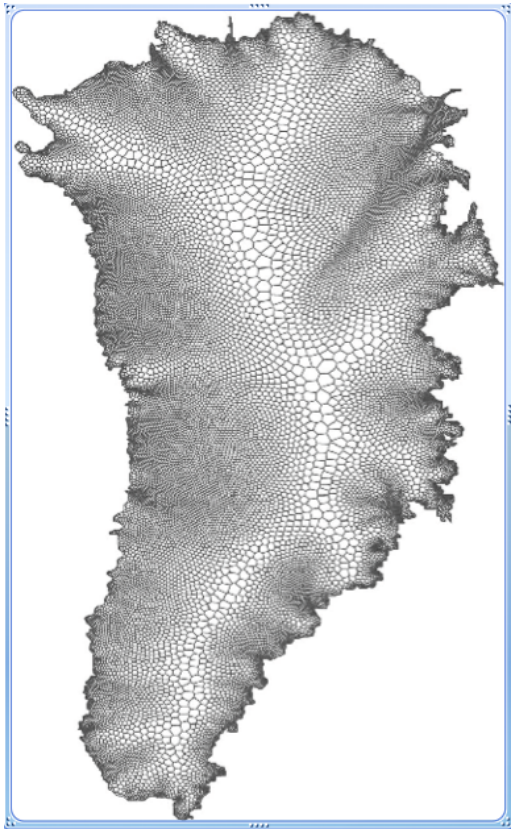
EXAMPLE: California Ocean Grid



A grid which is focussed for study of ocean currents near California.

EXAMPLE: Greenland Ice Sheet Grid





Estimated speed of Greenland ice sheet; a corresponding CVT grid.

CONCLUSION: References

- Qiang Du, Vance Faber, Max Gunzburger, *Centroidal Voronoi Tessellations: Applications and Algorithms,* SIAM Review, Volume 41, Number 4, December 1999, pages 637-676.
- Per-Olof Persson, Gilbert Strang,¡br¿ *A Simple Mesh Generator in MATLAB,* SIAM Review, Volume 46, Number 2, June 2004, pages 329-345.
- Robert Renka, *Algorithm 772: STRIPACK: Delaunay Triangulation and Voronoi Diagram on the Surface of a Sphere,* ACM Transactions on Mathematical Software, Volume 23, Number 3, September 1997, pages 416-434.
- Todd Ringler, Lili Ju, Max Gunzburger, *A multiresolution method for climate system modeling: application of spherical centroidal Voronoi tessellations*, Ocean Dynamics, Volume 58, Number 5-6, 2008, pages 475-498.
- Jonathan Shewchuk, *Delaunay Refinement Algorithms for Triangular Mesh Generation,* Computational Geometry, Theory and Applications, Volume 23, May 2002, pages 21-74.