The goal of this project is to implement basic vector visualizations (glyphs, streamlines, and Line Integral Convolution or LIC), as well as colormapping of derived scalar attributes (vector magnitude and vorticity). There is a set of tasks for which you submit images and explanations, all with specific filenames, identified below. You will create SVG files that contain raster images you create as well as line segments that you compute (for isocontours, glyphs, and streamlines). You will need to use (and adapt) your code from Project 2 for probing a field at arbitrary locations in world space; and from Project 3 for doing isocontours.

The Project 4 Tips page has useful information; also watch the scivis-2013 mailing list.

Your CNetID-scivis-2013 directories have been populated with a proj4 directory; do an svn update. Everything related to Project 4 is to be added to the proj4 directory and nowhere else. Images and text files belong in proj4, not proj4/code. The proj4/code subdirectory is where you hand in your code, and only code.

If you are working individually, you can ignore this paragraph. If working in a pair, and you are more than welcome to do so, both students should hand in a little two-line plain text file, named 00-pair.txt, in the proj4 directory. 00-pair.txt lists the names and CNetID@uchicago.edu emails of both members of the pair, one per line, as follows:

Jane H. H. Addams, jhhadams@uchicago.edu
Daniel H. Burnham, nolittleplans@uchicago.edu

The first student listed is also responsible for submitting all the completed work. We will only be grading the work submitted under the first student’s repository, i.e. in the jhhadams-scivis-2013 repository in this example. The second student must hand the same 00-pair.txt file. The graders will ignore any other files submitted by the second student.

New Data:

As with the last project, you will find new materials for this project in the scivis-2013 SVN repository described here. An svn update inside your checkout of scivis-2013 should create new files, described below. Assume that all the images are using the same orthonormal basis $\mathcal{B}$ for world-space as in Project 2 (first basis vector $b_1$ points right, second basis vector $b_2$ points up). Assume also that the vector-valued images contain vectors which are themselves expressed in the same basis $\mathcal{B}$.

- **elev/everesta{.png,-orient.txt}, elev/everestahe{.png,-orient.txt}, elev/everestasmhe{.png,-orient.txt}**: Oriented elevation (scalar) images of the Everest area. The “he” images have been histogram-equalized to better use the available 16-bits of the PNG image, to increase the accuracy of the gradients (especially in the nearly flat areas in the Ganges). The “everestasmhe” image has been smoothed significantly to make the computed flowlines simpler and more predictable.

- **vecimg/testvec{0,1,2,3,4,5,6,7}.txt**: test vector datasets, written as text files in a simple format. A point $p$, with coordinates $[p]_\mathcal{B} = \begin{bmatrix} x \\ y \end{bmatrix}$, is in the domain of these fields if both $x$ and $y$ are in $[-1, 1]$. The following equations describe the interior of the fields (approx $x, y \in [-0.62, 0.62]$); the magnitude of the vector field smoothly goes to 0 at the domain boundary. $v_i$ is stored in vecimg/testveci.

---

1The basis for the vector values could be entirely different, in part because of a difference of units between positions and vectors, but correctly handling this complication doesn’t really make the project more educational.
\[
[v_0(p)]_B = \begin{bmatrix} x \\ y \end{bmatrix}, \quad [v_1(p)]_B = \begin{bmatrix} -x \\ -y \end{bmatrix}, \quad [v_2(p)]_B = \begin{bmatrix} -x \\ y \end{bmatrix}, \quad [v_3(p)]_B = \begin{bmatrix} x \\ -y \end{bmatrix}, \quad [v_4(p)]_B = \begin{bmatrix} y \\ x \end{bmatrix}, \quad [v_5(p)]_B = \begin{bmatrix} -y \\ -x \end{bmatrix}, \quad [v_6(p)]_B = \begin{bmatrix} -y \\ x \end{bmatrix}, \quad [v_7(p)]_B = \begin{bmatrix} y \\ -x \end{bmatrix}
\]

- vecimg/turb2d.txt: 2D slice of a turbulent flow simulation. The domain of the field is points \((x, y)\) with \(x \in [0.0, 6.85]\) and \(y \in [-1.866, 1.866]\)
- vecimg/read2vecs.py: utility function for reading the vector fields. Read through this function to understand the format and how it is being parsed.

**What to implement:**

All the tasks require you to numerically solve a differential equation of the form

\[
\frac{dp(t)}{dt} = v(p(t))
\]

to determine a path \(p(t)\) through the vector field \(v(x)\) being visualized. The goal is to find a path \(p(t)\) that is everywhere tangent to the vector field. Because tangency with a vector doesn’t depend on the vector length, we could also define a different vector field of normalized vectors

\[
\hat{v}(x) = \frac{v(x)}{|v(x)|}
\]

and then solve

\[
\frac{dp(t)}{dt} = \hat{v}(p(t))
\]

Normalizing the vectors simplifies how we choose a numeric step size because the solution no longer depends on the magnitude \(|v(x)|\). With some choice of step size \(h\), you will implement Euler integration:

\[
p_{n+1} = p_n + h\hat{v}(p_n)
\]

or the Midpoint method (sometimes called Runge-Kutta order 2 or RK2):

\[
p_{n+1} = p_n + h\hat{v}(p_n + \frac{h}{2}\hat{v}(p_n)).
\]

Only students in 33710 will be expected to implement RK2.

For computing streamlines, the vector field \(v\) is the given vector field. For computing flowlines through a scalar field, \(v\) is the negative gradient of the scalar field: the flowline goes downhill, against the height gradient. In both cases, the computation begins at a seed or initialization point, and then proceeds in two opposite directions; the two halves are joined together to create the streamline or flowline path. By computing along both downstream (integrating along \(v\)) and upstream (integrating along \(-v\)) directions, the computed path visualizes the field in a neighborhood around the seedpoint. Whether integrating along \(v\) or \(-v\), the integration must stop when the position steps outside the field domain. Another useful termination criterion is to stop integration after some fixed number of steps.

**The Tasks:**

Pay attention to the list of files to hand in for each task, and the exact filenames of the requested files.

1. “flow”: The goal of this task is to make sure that you can integrate (3) in a case where bugs will be visually obvious: flowlines through a scalar field must be perpendicular to any isocontours they intersect. You will compute Marching Squares isocontours and flowlines from the elev/everestasmhe.png image (note the smhe), with orientation elev/everestasmhe-orient.txt. There should be 7 isocontours, at values:
There should be 6 flowlines, seeded at world-space positions \((x, y)\)

\[
(49.8,315.0) (167.0,315.0) (141.6,47.0) (274.4,259.0) (399.4,259.0) (422.8,407.0)
\]

From these starting locations, you should be able to compute flowlines showing the approximate course of the Ganges, and how rain near Mount Everest (if it somehow rained there) would flow into the Ganges. Use Euler integration with bilinear interpolation (convolution with ‘tent’ and its derivative), integrating along the normalized gradient, as with Eq. (3), going both upstream and downstream. Try a stepsize of about \(h = 0.5\), going for at most 1000 steps. Draw the isocontours and the flowlines into an SVG file (as in Project 3), using a colormap of the non-smoothed image `elev/everesta.png` (with orientation `elev/everesta-orient.txt`) as a backdrop. The quantization range in `elev/everesta.png` is the same as `elev/everest.png` in Project 1, so your experience colormapping that will be relevant. Make the flowlines and isocontours thin enough so that their shape is clear, and with a color that helps them stand out from the colormapped image behind it. The flowline and isocontour colors can be constant. The general goal of the image colormap is to use increasing luminance for increasing elevation, without using any highly saturated colors (which would detract from the isocontour and flowline visability).

What to hand in:

(a) `flow-cmap.txt`: Colormap used on `everesta.png`.

(b) `flow-img.png`: Result of colormapping `everesta.png`.

(c) `flow.svg`: SVG file showing isocontours and flowlines, computed from `elev/everestasmhe.png`, but drawn over over `flow-img.png`.

2. “glyphs”: The goal is to compute short streamlines with arrow heads in the `vecimg/testveci` vector fields. With some step size \(h\) you will integrate Eq. (1) for some fixed number of steps \(N\), using Euler integration. By integrating Eq. (1) instead of (3), the length of the computed streamline becomes a visual representation of the local vector field magnitude. Draw an arrow head on the downstream end of the streamline, so that it looks like a curving arrow showing the direction of flow. The arrow head can be as simple as two additional line segments starting at the streamline end. It will look nicer to scale the arrow head according to vector magnitude, but that is optional. The 16 streamline seedpoints should be (in worldspace) \((\pm 0.6, 0)\), \((0, \pm 0.6)\), \((\pm 0.6, \pm 0.6)\) \((\pm 0.3, \pm 0.3)\), \((\pm 0.15, 0)\), \((0, \pm 0.15)\). You should choose \(h\), \(N\), the arrow head size, and a line thickness, so that the streamlines create an uncluttered depiction of the field structure. Draw the streamlines into SVG files, with no background image.

What to hand in:

(a) `glyph0.svg, glyph1.svg, ... glyph7.svg`: 8 different SVG files, where `glyphi.svg` is the visualization of `testveci.txt`.

(b) `glyph-info.txt`: What \(h\) and \(N\) did you use, and how did you choose these values?

3. “evrk2”: (for 33710 students only) In vector field `testvec6.txt` (note the 6), demonstrate the difference between Euler and RK2 integration. Compute two streamlines, both seeded at \((0.4, 0)\), both drawn with an arrowhead at the end, both using some value of \(h\) and \(N\) which you pick for this task, but with one streamline using Euler, and the other streamline using RK2. Use two different colors for the two streamlines. Your choice of \(h\) and \(N\) should result in a figure that clearly highlights the difference in the two integration methods.

What to hand in:

(a) `evrk2.svg`: The diagram you created to depict the difference between Euler and RK2 integration.

(b) `evrk2-info.txt`: What \(h\) and \(N\) did you use, and how did you choose these values?

4. “turb”: The goal here is to visualize the `vecimg/turb2d.txt` dataset by drawing streamlines integrating Eq. (1) over a background image colored by vector field vorticity (2D curl). How you do this is largely up to; the choices to make include: how many and where to put the streamline seed locations, the streamline
integration parameters, how to display the streamlines in the SVG file, and how to colormap the underlying image of vorticity. 33710 students may use Euler integration for prototyping, but their final visualization should use RK2 integration.

What to hand in:

(a) turb-img.png: The colormapped image of vector field vorticity that you compute. It should have the same resolution and assume the same orientation as the turb2d.txt vector field. This way you do not have to do mappings from world-space to index-space to create this image; the derivatives are only on the discrete grid of turb2d.txt. No additional description of the colormap (of vorticity) is needed; it will be clear from the image.

(b) turb.svg: The visualization of streamlines drawn over turb-img.png that you create.

(c) turb-info.txt: What choices did you make to create this visualization, and what informed your choice?

In addition to the files listed above, hand in all your code in the proj4/code directory. Nothing besides your code should be in this directory. You need to submit the code that you had to write from scratch for this project; you are also welcome to submit code from previous projects that you used or edited for this project.