Projects for

Chapter 5: Scalar Visualization

1 PROJECT 1

Using the code model for the rainbow colormap function (Listing 5.1), design and write implementations for the colormaps illustrated in Figure 5.2. That is, for each colormap,

- Write a function c having the same signature as the one indicated in Listing 5.1;
- Implement the function body to return the colors displayed in the color legends in Figure 5.2.

Test your functions by creating a simple visualization application that takes as input a vector of $N = 100$ scalar values $v_i, 1 \leq i \leq N$ uniformly distributed between 0 and 1, and displays as output a color legend constructed by drawing N small rectangles r_i, each being colored by a color $c_i = c(v_i)$ obtained by applying your colormap function c to the respective scalar value v_i. Compare your results with those shown in Figure 5.2.

Hints: Note that all colormaps used in Figure 5.2 are obtained by linear interpolation between two or three basic colors. Hence, first decide which are these basic colors for each case. Next, use linear interpolation to construct the colormap function c.

2 PROJECT 2

Implement a simple visualization application that loads (or creates) a scalar dataset on a 2D uniform grid, and next applies several of the colormaps constructed in Project 1 to the cells of the grid, using bilinear scalar interpolation. The resulting visualization should show the grid cells, colored by the interpolated scalars, as well as the color legend for the currently-selected color map. Next, implement an interactive querying facility that *links* the color legend and the color-mapped grid in two directions:

1. Brushing with the mouse over the color-mapped grid: Find the actual point (x, y) under the mouse; determine its interpolated scalar value; next, highlight the respective value in the color legend, and draw the actual numerical scalar value as a label close to the highlighted value.

2. Brushing with the mouse over the color legend: Find the actual scalar value v under the mouse; next, find all vertices (x, y) in the grid that have that value, and highlight them, *e.g.* by coloring them with a reserved color not present in the colormap.

Hints:

- For the brushing mode 1 above, first convert the mouse coordinates into grid (or world) coordinates. Next, determine the cell in which these world coordinates fall, and the corresponding cell-coordinates (also called parametric coordinates). Finally, use the parametric coordinates to find the value at the point under the mouse by interpolating the values at the cell vertices.

- For the brushing mode 2 above, first decide how to select points in the grid that have a given scalar value s. You can do this at increasingly accurate levels of detail:

  - Select cells which contain at least one point having value s (by comparing cell-vertex values with s);

  - Scan all pixels of all cells and select those which show the value s;

  - Compute and display a contour line (isoline) for value s.

3 PROJECT 3

Given a scalar dataset sampled on a 2D uniform or rectilinear grid, implement the *marching squares* algorithm to render one or several isolines I_i for a set of given scalar values v_i. The algorithm is described in Section 5.3.1. Proceed as follows:

1. Use the sample code provided for Chapter 5 to implement the 2D scalar dataset;

2. Fill in the dataset by *e.g.* sampling a two-variable function $f(x, y)$;

3. For each isoline value v_i

    - Initialize I_i to the empty set;
    - Construct codes for all dataset vertices *vs* the value v_i;
    - For each dataset cell c_j, construct an isoline fragment f_j following the cases in Figure 5.12, and next add f_j to the set I_i;
    - Render I_i by drawing it as a polyline.

    Test your program by saving your 2D scalar dataset to the VTK *uniform grid* format, loading it into ParaView, and creating isolines for the same values in ParaView. The resulting ParaView visualization should be identical to your own visualization. Creating isolines in ParaView is demonstrated in the samples included for Chapter 5.

4 PROJECT 4

Given a scalar dataset sampled on a 3D uniform grid with cubic cells, implement the *dividing cubes* algorithm to render an isosurface I for a given scalar value v. The algorithm is described in Section 5.3.2. Proceed as follows:

1. Extend the sample code provided for Chapter 3 (implementation of 2D uniform grids) to implement a 3D uniform grid. Alternatively, you can use the sample code provided for Chapter 10, which directly gives you a complete, albeit more complex, implementation of 3D uniform grids.
2. Fill in the dataset by *e.g.* sampling a three-variable function $f(x, y, z)$.
3. Create an empty set of cells C, which will store the voxel approximation of the isosurface.
4. Implement a recursive function f that tests each dataset cell c against intersection with the isosurface I by checking its vertex values against the isovalue v. If c intersects I, f should recursively test all eight cells obtained by uniformly subdividing c along the midpoints of its edges. When a maximum recursion depth (set by the user) is reached, f should add c to the set C.
5. Call f to test all cells of the input dataset.
6. Render the resulting cells in C *e.g.* by drawing small cubes or drawing their center

points. Alternatively, you can use the high-quality, but more complex, splat-based voxel rendering techniques provided in the samples for Chapter 10.

End of Projects for Chapter 5: Scalar Visualization