

Lab 5 报告

蒋锦豪 2400011785

2025 年 12 月 27 日

Task 1: Parallel Coordinates Visualization

结构体 `CoordinateStates` 主要包含以下部分:

首先对读入的数据 `data` 进行预处理和存储, 将数据保存在二维数组中, 并且记录各字段的最小值 `minData` 和最大值 `maxData`. 这部分的代码如下:

```
1 std::vector<std::string>          labelName = { "cylinders", "displacement", "horsepower", "weight", "acceleration", "mileage", "year" };
2 std::vector<int>                  labelIdx  = { 0, 1, 2, 3, 4, 5, 6 };
3 int                                mainIdx   = 0; // main axis index
4 int                                dataNum   = 7, dataSize;
5 std::vector<std::vector<float>> dataTable;
6 std::vector<float>                minData, maxData;
7 CoordinateStates(std::vector<Car> const& data) {
8     dataSize = data.size();
9     for (auto car : data) {
10         std::vector<float> carData;
11         carData.push_back(car.cylinders);
12         ...
13         dataTable.push_back(carData);
14     }
15     minData = dataTable[0];
16     maxData = dataTable[0];
17     for (size_t i = 0; i < dataSize; ++i) {
18         for (size_t j = 0; j < dataNum; ++j) {
19             minData[j] = std::min(dataTable[i][j], minData[j]);
20             maxData[j] = std::max(dataTable[i][j], maxData[j]);
21         }
22     }
23 }
```

然后通过线性插值的方法形成各数据点在绘图区域的坐标, 这部分的代码如下:

```

1 std::vector<std::vector<std::pair<float, float>>> dataPlotPos;
2 const float minY = 0.15f, maxY = 0.85f;
3 void ConstructPlotPos() {
4     dataPlotPos.clear();
5     for (size_t i = 0; i < dataSize; ++i) {
6         std::vector<std::pair<float, float>> line;
7         for (int j = 0; j < dataNum; ++j) {
8             int k = labelIdx[j];
9             float yPos = minY + (maxData[k] - dataTable[i][k]) / (maxData[k] - minData[k]) * (maxY - minY);
10            line.push_back(std::pair<float, float>((2 * j + 1) / (2.0f * dataNum), yPos));
11        }
12        dataPlotPos.push_back(line);
13    }
14 }
```

然后是绘图函数 `Paint` 的实现. 根据生成的坐标列表 `dataPlotPos` 按照主轴 `mainIdx` 的大小进行排序, 然后用预先设置的两种颜色 `red` 和 `blue` 进行插值得到数据对应的折线的颜色; 随后给各坐标轴进行矩形框高亮, 对主轴使用红色, 对其余轴使用灰色; 最后在各坐标轴上下标出数据和文本作为注释. 这部分的代码如下:

```

1 const float yShift = 0.05f, rectWidth = 0.015f, axisWidth = 2.0f, frameWidth = 1.5f;
2 const float textShift = 0.025f, lineHeight = 0.01f;
3 void Paint(Common::ImageRGB & input) {
4     ConstructPlotPos();
5     int idx = mainIdx;
6     std::sort(dataPlotPos.begin(), dataPlotPos.end(), [idx](const std::vector<std::pair<float, float>> &a, const std::vector<std::pair<float, float>> &b) {
7         return a[idx].second < b[idx].second;
8     });
9
10    SetBackGround(input, backwhite);
11    for (size_t i = 0; i < dataSize; ++i) {
12        float s = 1.0f * i / dataSize;
13        float t = 0.5 + 4 * std::pow(s - 0.5, 3);
14        glm::vec4 curColor = t * blue + (1 - t) * red;
15        glm::vec2 from(dataPlotPos[i][0].first, dataPlotPos[i][0].second), to;
16        for (size_t j = 1; j < dataNum; ++j) {
17            to = glm::vec2(dataPlotPos[i][j].first, dataPlotPos[i][j].second);
18            DrawLine(input, curColor, from, to, 0.1f);
19            from = to;
```

```

20     }
21 }
22 for (size_t i = 0; i < dataNum; ++i) {
23     float centerX = (i * 2 + 1) * 1.0f / (dataNum * 2);
24     glm::vec2 leftTop(centerX - rectWidth / 2, minY - yShift), size(rectWidth, maxY - minY + 2 * yShift);
25     DrawLine(input, darkgray, glm::vec2(centerX, minY - yShift), glm::vec2(centerX, maxY + yShift), 2.0f);
26     DrawFilledRect(input, (i == idx) ? backred : lightgray, leftTop, size);
27     DrawRect(input, frontwhite, leftTop, size, frameWidth);
28 }
29 for (size_t i = 0; i < dataNum; ++i) {
30     float centerX = (i * 2 + 1) * 1.0f / (dataNum * 2);
31     PrintText(input, black, glm::vec2(centerX, minY - yShift - textShift * 2), lineHeight,
32     labelName[labelIdx[i]]);
33     PrintText(input, black, glm::vec2(centerX, minY - yShift - textShift), lineHeight, std::
34     to_string((int) maxData[labelIdx[i]]));
35     PrintText(input, black, glm::vec2(centerX, maxY + yShift + textShift), lineHeight, std::
36     to_string((int) minData[labelIdx[i]]));
37 }
38 }

```

最后是对鼠标交互的处理。基本的设计思路是左键单击轴可以改变主轴，右键依次单击两个不同的轴可以将两个轴进行交换。在右键单击第一个轴时，用 `labelSwap` 记录是否正在执行交换操作。同时，在交换完成后，需要判断主轴是否被交换，若是则需要同时修改主轴的位置。具体的代码实现如下：

```

1 const float clickShift = 0.06f;
2 bool labelSwap = false;
3 glm::vec2 startPos, endPos;
4 int startIdx = -1, endIdx = -1;
5 void ResetLabelSwap() {
6     startIdx = -1;
7     endIdx = -1;
8     labelSwap = false;
9 }
10
11 bool Update(InteractProxy const & proxy) {
12     if (!proxy.IsHovering()) return false;
13     if (proxy.IsClicking()) {
14         glm::vec2 clickPos = proxy.MousePos();
15         for (size_t i = 0; i < dataNum; ++i) {

```

```

16     float centerX = (i * 2 + 1) * 1.0f / (dataNum * 2);
17     if (std::abs(clickPos.x - centerX) < clickShift && clickPos.y > (minY - clickShift) &&
18     clickPos.y < (maxY + clickShift)) {
19         mainIdx      = i;
20         return true;
21     }
22 }
23 if (proxy.IsClicking(false)) {
24     if (!labelSwap) {
25         startPos = proxy.MousePos();
26         for (size_t i = 0; i < dataNum; ++i) {
27             float centerX = (i * 2 + 1) * 1.0f / (dataNum * 2);
28             if (std::abs(startPos.x - centerX) < clickShift && startPos.y > (minY - clickShift)
29             && startPos.y < (maxY + clickShift)) {
30                 startIdx = i;
31             }
32             if (startIdx != -1) labelSwap = true;
33             return false;
34 } else {
35         endPos   = proxy.MousePos();
36         for (size_t i = 0; i < dataNum; ++i) {
37             float centerX = (i * 2 + 1) * 1.0f / (dataNum * 2);
38             if (std::abs(endPos.x - centerX) < clickShift && endPos.y > (minY - clickShift) &&
39             endPos.y < (maxY + clickShift)) {
40                 endIdx = i;
41             }
42             if (endIdx != -1 && startIdx != endIdx) {
43                 if (startIdx == mainIdx) mainIdx = endIdx;
44                 if (endIdx == mainIdx) mainIdx = startIdx;
45                 std::swap(labelIdx[startIdx], labelIdx[endIdx]);
46                 ResetLabelSwap();
47                 return true;
48 } else {
49                 ResetLabelSwap();
50                 return false;
51 } }
52 }
```

```
53     }
54 }
```

在 `PaintParallelCoordinates` 函数中实现所有功能的代码如下:

```
1 bool PaintParallelCoordinates(Common::ImageRGB & input, InteractProxy const & proxy, std::vector<Car
2 > const & data, bool force) {
3     static CoordinateStates states(data); // initialize
4     bool change = states.Update(proxy); // update according to user input
5     if (!force && !change) return false; // determine to skip repainting
6     states.Paint(input); // visualize
7 }
```

最终实现的效果如下所示:

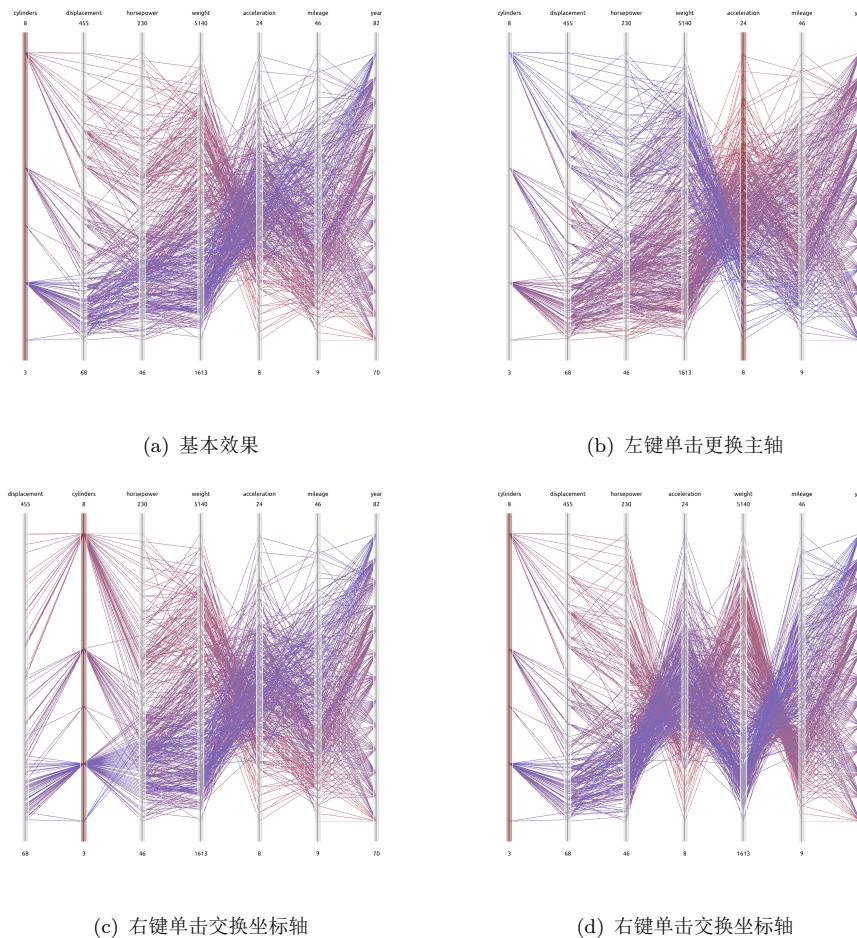


图 1: Parallel Coordinates Visualization

Task 2: Flow Visualization

按照 LIC 的原理进行采样-卷积, 代码如下所示:

```

1 void LIC(ImageRGB & output, Common::ImageRGB const & noise, VectorField2D const & field, int const &
2   step) {
3   // your code here
4   int width = output.GetSizeX(), height = output.GetSizeY();
5   float x, y, dt_x, dt_y, dt;
6   float forward_total, backward_total, weight;
7   glm::vec3 forward_sum, backward_sum;
8   for (int i = 0; i < width; ++i) {
9     for (int j = 0; j < height; ++j) {
10       x = i;
11       y = j;
12       forward_sum = glm::vec3(0);
13       forward_total = 0;
14       for (int k = 0; k < step; ++k) {
15         glm::vec2 v = field.At((int) x, (int) y);
16         dt_x = 0;
17         dt_y = 0;
18         if (v.x > 0) {
19           dt_x = ((std::floor(x) + 1) - x) / v.x;
20         } else if (v.x < 0) {
21           dt_x = (x - (std::ceil(x) - 1)) / (-v.x);
22         }
23         if (v.y > 0) {
24           dt_y = ((std::floor(y) + 1) - y) / v.y;
25         } else if (v.y < 0) {
26           dt_y = (y - (std::ceil(y) - 1)) / (-v.y);
27         }
28         if (v.x == 0 && v.y == 0) dt = 0;
29         else dt = std::min(dt_x, dt_y);
30         x = std::min(std::max(x + v.x * dt, 0.0f), width - 1.0f);
31         y = std::min(std::max(y + v.y * dt, 0.0f), height - 1.0f);
32         weight = std::pow(std::cos(0.46 * k), 2);
33         forward_sum += noise.At((int) x, int(y)) * weight;
34         forward_total += weight;
35       }
36       x = i;
37       y = j;

```

```
37     backward_sum    = glm::vec3(0);
38     backward_total = 0;
39     for (int k = 1; k < step; ++k) {
40         glm::vec2 v = (-1.0f) * field.At((int) x, (int) y);
41         dt_x        = 0;
42         dt_y        = 0;
43         if (v.x > 0) {
44             dt_x = ((std::floor(x) + 1) - x) / v.x;
45         } else if (v.x < 0) {
46             dt_x = (x - (std::ceil(x) - 1)) / (-v.x);
47         }
48         if (v.y > 0) {
49             dt_y = ((std::floor(y) + 1) - y) / v.y;
50         } else if (v.y < 0) {
51             dt_y = (y - (std::ceil(y) - 1)) / (-v.y);
52         }
53         if (v.x == 0 && v.y == 0) dt = 0;
54         else dt = std::min(dt_x, dt_y);
55         x      = std::min(std::max(x + v.x * dt, 0.0f), width - 1.0f);
56         y      = std::min(std::max(y + v.y * dt, 0.0f), height - 1.0f);
57         weight = std::pow(std::cos(-0.46 * k), 2);
58         backward_sum += noise.At((int) x, int(y)) * weight;
59         backward_total += weight;
60     }
61     output.At(i, j) = (forward_sum + backward_sum) / (forward_total + backward_total);
62 }
63 }
64 }
```

实现的效果图如下：

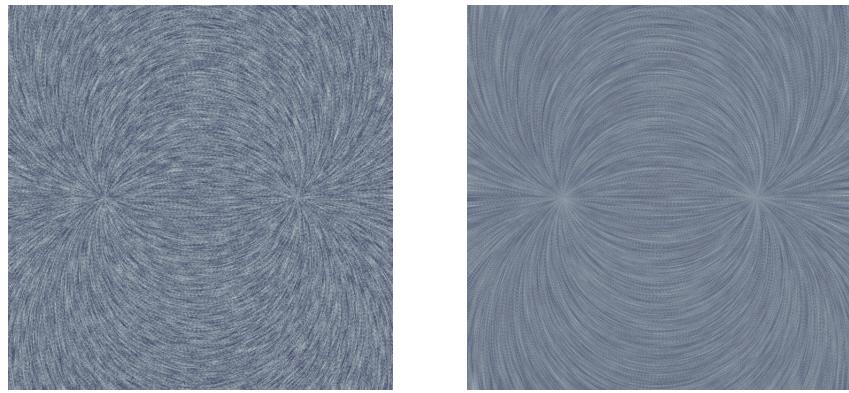


图 2: 对 Bipole 流场的可视化

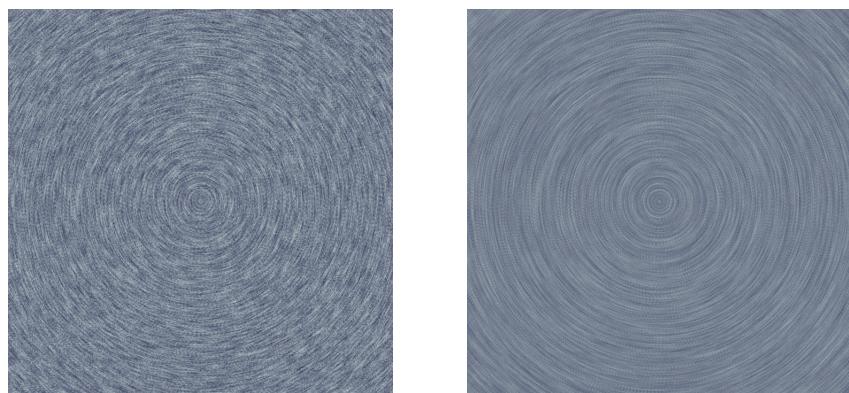


图 3: 对 Circle 流场的可视化

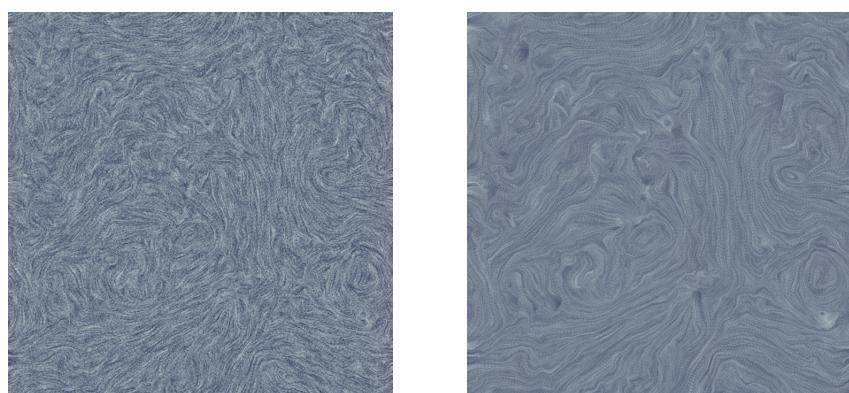


图 4: 对 Turbulence 流场的可视化