Lab 5 报告

蒋锦豪 2400011785

2025 年 12 月 8 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:

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

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:

```
const float yShift = 0.05f, rectWidth = 0.015f, axisWidth = 2.0f, frameWidth = 1.5f;
const float textShift = 0.025f, lineHeight = 0.01f;
void Paint(Common::ImageRGB & input) {
    ConstructPlotPos();
    int idx = mainIdx;
    std::sort(dataPlotPos.begin(), dataPlotPos.end(), [idx](const std::vector<std::pair<float, float>> & a, const std::vector<std::pair<float, float>> & b) {
        return a[idx].second < b[idx].second;
    });

    SetBackGround(input, backwhite);
    for (size_t i = 0; i < dataSize; ++i) {
        float     s        = 1.0f * i / dataSize;
        float     t        = 0.5 + 4 * std::pow(s - 0.5, 3);
        glm::vec4 curColor = t * blue + (1 - t) * red;
        glm::vec2 from(dataPlotPos[i][0].first, dataPlotPos[i][0].second), to;
        for (size_t j = 1; j < dataNum; ++j) {
            to = glm::vec2(dataPlotPos[i][j].first, dataPlotPos[i][j].second);
            DrawLine(input, curColor, from, to, 0.1f);
            from = to;
```

```
        }
    }
    for (size_t i = 0; i < dataNum; ++i) {
        float     centerX = (i * 2 + 1) * 1.0f / (dataNum * 2);
        glm::vec2 leftTop(centerX - rectWidth / 2, minY - yShift), size(rectWidth, maxY - minY + 2 *
     yShift);
        DrawLine(input, darkgray, glm::vec2(centerX, minY - yShift), glm::vec2(centerX, maxY +
    yShift), 2.0f);
        DrawFilledRect(input, (i == idx) ? backred : lightgray, leftTop, size);
        DrawRect(input, frontwhite, leftTop, size, frameWidth);
    }
    for (size_t i = 0; i < dataNum; ++i) {
        float centerX = (i * 2 + 1) * 1.0f / (dataNum * 2);
        PrintText(input, black, glm::vec2(centerX, minY - yShift - textShift * 2), lineHeight,
    labelName[labelIdx[i]]);
        PrintText(input, black, glm::vec2(centerX, minY - yShift - textShift), lineHeight, std::
    to_string((int) maxData[labelIdx[i]]));
        PrintText(input, black, glm::vec2(centerX, maxY + yShift + textShift), lineHeight, std::
    to_string((int) minData[labelIdx[i]]));
    }
}
```

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:

```
const float clickShift = 0.06f;
bool        labelSwap  = false;
glm::vec2   startPos, endPos;
int         startIdx = -1, endIdx = -1;
void ResetLabelSwap() {
    startIdx  = -1;
    endIdx    = -1;
    labelSwap = false;
}

bool Update(InteractProxy const & proxy) {
    if (! proxy.IsHovering()) return false;
    if (proxy.IsClicking()) {
        glm::vec2 clickPos = proxy.MousePos();
        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) &&
    clickPos.y < (maxY + clickShift)) {
18                  mainIdx      = i;
19                  return true;
20              }
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)
    && startPos.y < (maxY + clickShift)) {
29                      startIdx = i;
30                  }
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) &&
    endPos.y < (maxY + clickShift)) {
39                      endIdx = i;
40                  }
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  }
```

上述几个部分就是结构体 `CoordinateStates` 的内容. 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
        > const & data, bool force) {
2       // your code here
3       // for example:
4       //   static CoordinateStates states(data);
5       //   SetBackGround(input, glm::vec4(1));
6       //   ...
7       static CoordinateStates states(data);                   // initialize
8       bool                    change = states.Update(proxy); // update according to user input
9       if (! force && ! change) return false;                  // determine to skip repainting
10      states.Paint(input);                                    // visualize
11      return true;
12  }
```