Lab5 Report

2300012929 尹锦润

Task 1

实现思路

按照说明提示，这是主函数思路：

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

也就是实现 init 函数、update 函数 和 paint 函数。

Init 函数

为了实现更加舒服，我们把数据全部变成 float 的 array 这样可以方便地通过循环进行绘图。

同时 `range_in_data` 表示的就是对应值的大小范围，而 `range_in_canvas` 就是绘图的框定区域（$\subseteq [\text{upperMargin}, \text{lowerMargin}]$。

```
CoordinateStates(std::vector<Car> const & data) {
    painted = false;
    selectedId = -1;
    for(int i = 0; i < 7; i++) {
        range_in_data[i][0] = std::numeric_limits<float>::max();
        range_in_data[i][1] = std::numeric_limits<float>::min();
    }
    for(auto v : data) {
        num_data.push_back({static_cast<float>(v.cylinders), v.displacement,
v.weight, v.horsepower, v.acceleration, v.mileage, static_cast<float>(v.year)});
        for(int i = 0; i < 7; i++) {
            range_in_data[i][0] = std::min(range_in_data[i][0], num_data.back()
[i]);
```

```
            range_in_data[i][1] = std::max(range_in_data[i][1], num_data.back()
[i]);
        }
    }
    for(int i = 0; i < 7; i++) range_in_canvas[i][0] = upperMargin,
range_in_canvas[i][1] = lowerMargin;
}
```

Update 函数

首先我们设置区域：

- Axis 区域，当鼠标放在轴上面所对应的区域。
- Axis 周围的框定区域，当鼠标放在轴附近，我们可以拖动来框定显示范围，也可以通过点击进行重设范围。

鼠标 Icon 切换

```
if(proxy.IsHovering()) {
    int tmpAxis = getAxis(proxy.MousePos());
    if(tmpAxis != -1) ImGui::SetMouseCursor(ImGuiMouseCursor_Hand); // set to
resize
    else {
        tmpAxis = getRangeAxis(proxy.MousePos());
        if(tmpAxis != -1) ImGui::SetMouseCursor(ImGuiMouseCursor_ResizeNS); //
set to hand
        else ImGui::SetMouseCursor(ImGuiMouseCursor_Arrow); // set to default
    }
} else ImGui::SetMouseCursor(ImGuiMouseCursor_Arrow); // set to default
```

根据区域进行判断。

点击行为的处理

```
if(proxy.IsClicking()) {
    int tmpAxis = getAxis(proxy.MousePos());
    if(tmpAxis != -1) {
        selectedId = tmpAxis;
        return true;
    }
    tmpAxis = getRangeAxis(proxy.MousePos());
    if(tmpAxis == -1) return false;
    /* reset range */
```

```
    range_in_canvas[tmpAxis][0] = upperMargin;
    range_in_canvas[tmpAxis][1] = lowerMargin;
    selectedId = tmpAxis;
    return true;
}
```

通过 `selectedID` 来判断是根据哪个轴来染色。

如果点击了框定区域那么就是重设框定区域。

拖动行为处理

```
int tmpAxis = getAxis(proxy.DraggingStartPoint());
if(tmpAxis != -1) { // start drag the range box.
    float rangeDelta = (proxy.MouseDeltaPos()).y / 2;
    rangeDelta = std::min(rangeDelta, lowerMargin - range_in_canvas[tmpAxis]
[1]);
    rangeDelta = std::max(rangeDelta, upperMargin - range_in_canvas[tmpAxis]
[0]);
    range_in_canvas[tmpAxis][0] += rangeDelta;
    range_in_canvas[tmpAxis][1] += rangeDelta;
    selectedId = tmpAxis;
    return true;
}
tmpAxis = getRangeAxis(proxy.DraggingStartPoint());
if(tmpAxis == -1) return false;
// start selected the range box.
selectedId = tmpAxis;
float rangeLow = proxy.DraggingStartPoint().y;
float rangeHigh = proxy.MousePos().y;
if(rangeLow > rangeHigh) std::swap(rangeLow, rangeHigh);
range_in_canvas[tmpAxis][0] = std::max(rangeLow, upperMargin);
range_in_canvas[tmpAxis][1] = std::min(rangeHigh, lowerMargin);
return true;
```

如果是 Axis 区域，就是拖动范围，否则是这是范围。需要注意 `range_in_canvas` 的范围限制。

Paint 函数

画数据点

```
/* Paint Lines */
int checkAxis = selectedId == -1 ? 0 : selectedId;
```

```
for(auto v : num_data) {
    bool ok = true;
    for(int i = 0; i < 7 && ok; i++) {
        float centerpos = delta * i + leftMargin;
        float pos = (v[i] - range_in_data[i][0]) / (range_in_data[i][1] -
range_in_data[i][0]) * (lowerMargin - upperMargin) + upperMargin;
        if(pos > range_in_canvas[i][1] || pos < range_in_canvas[i][0]) ok =
false;
    }
    glm::vec2 lastpos;
    float rate = (v[checkAxis] - range_in_data[checkAxis][0]) /
(range_in_data[checkAxis][1] - range_in_data[checkAxis][0]);
    glm::vec4 color = rate * lightStartColor + (1 - rate) * lightEndColor;
    if(!ok) color = notInRangeColor;
    for(int i = 0; i < 7; i++) {
        float centerpos = delta * i + leftMargin;
        glm::vec2 pos(centerpos + rectSize.x / 2, (v[i] - range_in_data[i][0]) /
(range_in_data[i][1] - range_in_data[i][0]) * (lowerMargin - upperMargin) +
upperMargin);
        if(i != 0) {
            DrawLine(input, color, lastpos, pos, 0.00002);
        }
        lastpos = pos;
    }
}
```

首先计算 `ok` 也就是这条线是不是在框定范围内，颜色应该设置为什么。

如果是 ok 的，那么颜色根据预先 const 的颜色进行插值。

然后就是遍历，根据 `lastpos` 不断画线。

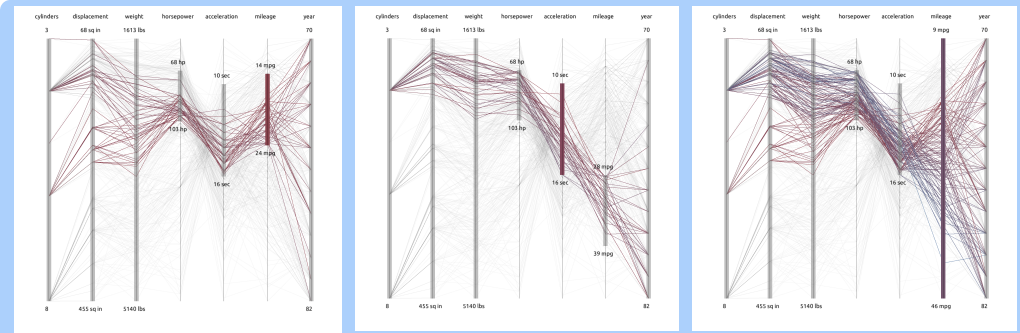
画框

```
/* Paint boxs */
for(int i = 0; i < 7; i++) {
    float centerpos = delta * i + leftMargin;
    glm::vec4 color = unselectedColor;
    if(i == selectedId) {
        float rate = (range_in_canvas[i][0] + range_in_canvas[i][1] - 2 *
upperMargin) / (2 * lowerMargin - 2 * upperMargin);
        color = rate * lightStartColor + (1 - rate) * lightEndColor;
        color.w = 1;
    }
    DrawFilledRect(input, color, glm::vec2(centerpos, range_in_canvas[i][0]),
glm::vec2(rectSize.x, range_in_canvas[i][1] - range_in_canvas[i][0]));
```

```
    DrawLine(input, lineColor, glm::vec2(centerpos + rectSize.x / 2,
upperMargin), glm::vec2(centerpos + rectSize.x / 2, lowerMargin), 0.005);
    PrintText(input, lineColor, glm::vec2(centerpos, upperMargin / 3),
upperMargin / 4, getCaption(i));
    int st = (range_in_canvas[i][0] - upperMargin) / (lowerMargin - upperMargin)
* (range_in_data[i][1] - range_in_data[i][0]) + range_in_data[i][0];
    int ed = (range_in_canvas[i][1] - upperMargin) / (lowerMargin - upperMargin)
* (range_in_data[i][1] - range_in_data[i][0]) + range_in_data[i][0];
    PrintText(input, lineColor, glm::vec2(centerpos, range_in_canvas[i][0] -
upperMargin / 4), upperMargin / 4, std::to_string(st) +getUnitName(i));
    PrintText(input, lineColor, glm::vec2(centerpos, range_in_canvas[i][1] +
upperMargin / 4), upperMargin / 4, std::to_string(ed) +getUnitName(i));
}
```

根据每个轴的范围绘制文本，以及相关矩形。矩形的颜色根据相关范围对应预先设置的颜色取 mid。

实现效果



实现了限制和拖动限制，根据选取的主轴来进行染色。

鼠标交互已经实现了，但是截图没有办法体现。

Task 2

实现思路

参考 python 代码和相关原理介绍。

我们每次遍历每个 particle 的图像。

对于每个 particle，我们向前 step 步，向后 step - 1 步，获取对应 particle 的位置，并且根据 noise 图像取点按照权重进行叠加。

```
for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) {
    float y = i, x = j;
    glm::vec3 forward_sum(0, 0, 0);
```

```
    float forward_total = 0;
    for (int k = 0; k < step; k++) {
        float dx = field.At(int(y), int(x)).x, dy = field.At(int(y), int(x)).y;
        float dt_x = 0, dt_y = 0, dt = 0;
        if (dy > 0) dt_y = (floor(y) + 1 - y) / dy;
        else if (dy < 0) dt_y = (y - (ceil(y) - 1)) / (-dy);
        if (dx > 0) dt_x = (floor(x) + 1 - x) / dx;
        else if (dx < 0) dt_x = (x - (ceil(x) - 1)) / (-dx);
        if (abs(dx) < eps && abs(dy) < eps) dt = 0;
        else dt = std::min(dt_x, dt_y);
        x = std::min(std::max(x + dx * dt, 0.f), n - 1.f);
        y = std::min(std::max(y + dy * dt, 0.f), m - 1.f);
        float weight = pow(cos(t + 0.46 * k), 2);
        forward_sum += noise.At(int(y), int(x)) * weight;
        forward_total += weight;
    }
    y = i, x = j;
    glm::vec3 backward_sum(0, 0, 0);
    float backward_total = 0;
    for (int k = 1; k < step; k++) {
        float dx = field.At(int(y), int(x)).x, dy = field.At(int(y), int(x)).y;
        dy *=- 1, dx *= -1;
        float dt_x = 0, dt_y = 0, dt = 0;
        if (dy > 0) dt_y = (floor(y) + 1 - y) / dy;
        else if (dy < 0) dt_y = (y - (ceil(y) - 1)) / (-dy);
        if (dx > 0) dt_x = (floor(x) + 1 - x) / dx;
        else if (dx < 0) dt_x = (x - (ceil(x) - 1)) / (-dx);
        if (abs(dx) < eps && abs(dy) < eps) dt = 0;
        else dt = std::min(dt_x, dt_y);
        x = std::min(std::max(x + dx * dt, 0.f), n - 1.f);
        y = std::min(std::max(y + dy * dt, 0.f), m - 1.f);
        float weight = pow(cos(t - 0.46 * k), 2);
        forward_sum += noise.At(int(y), int(x)) * weight;
        forward_total += weight;
    }
    output.At(i, j) = (forward_sum + backward_sum) / (forward_total +
backward_total);
}
```

实现效果