可视计算与交互概论-Lab5

xTryer

2024/12/6

1 Task1:Parallel Coordinates Visualization

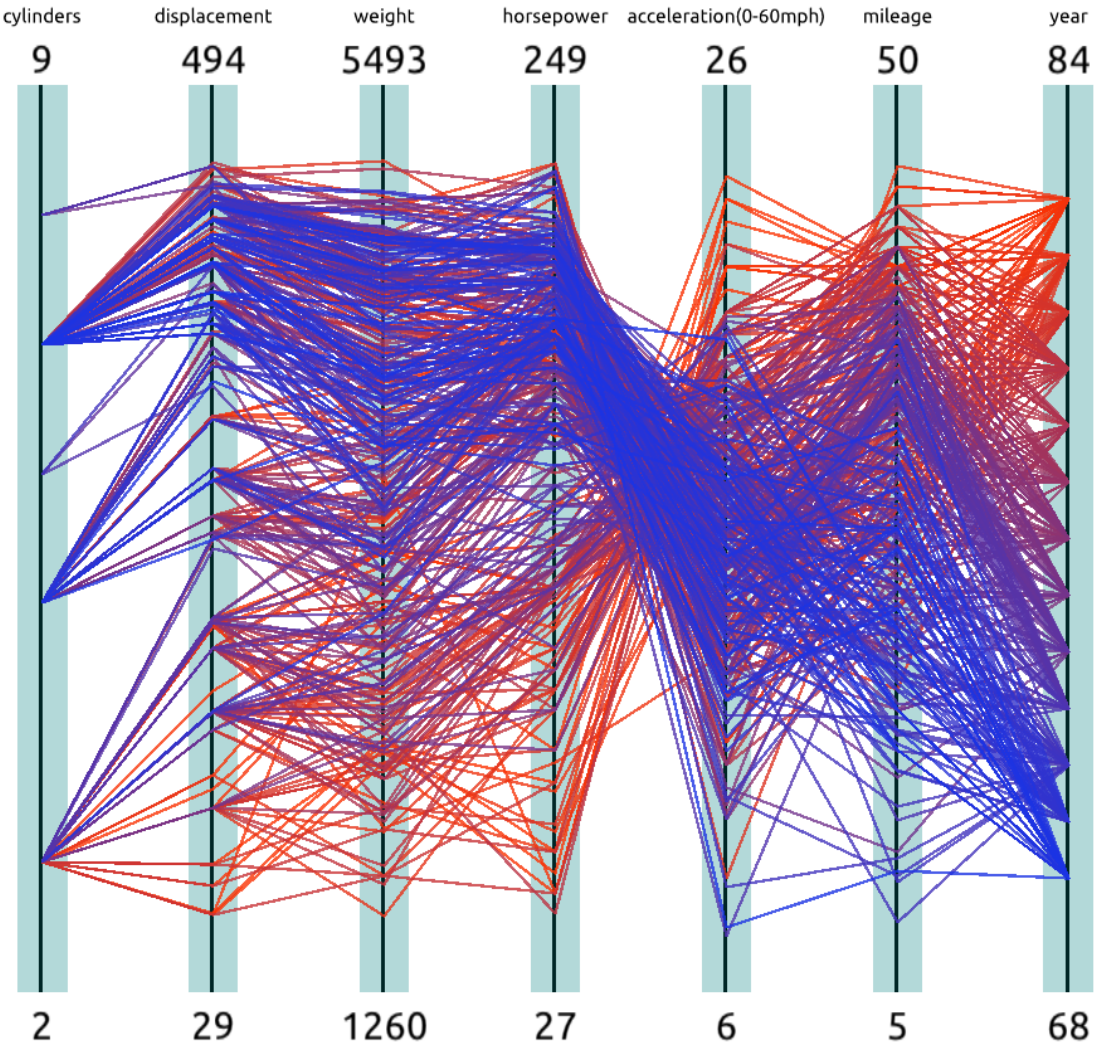
整体思路：

首先，在 CoordinatesStates 这个结构体中，我们读入数据，获得 data 在 cylinders, displacement,weight,horsepo mileage,year 这七个属性上的最大值与最小值，以供后续作图使用。

在 PaintParallelCoordinates 中，我们先确定代表每个维度的 parallel axe 的坐标位置，为了美观起见，可以增加 padding 使图表集中在画面中心区域，给四周留空。然后我们再根据给定的 data 来进行 polyline 的绘制，具体实现是利用插值公式来确定线条的起始位置和终止位置，插值函数如下:

```
float linear_interpolation(float val_,float min_, float max_, float down_, float up_)
{
    // down_ is the max_y in the pixel_coordinates // up_ is the min_y in the
    pixel_coordinates
    // min_ is the minvalue of the attribute //max_ is the maxvalue of the attribute
    return (val_ - min_) / (max_ - min_)*(down_-up_)+up_;
}
```

得到效果图如下：



进一步，我们考虑实现鼠标交互，主要实现四个功能:

- 1. 点击左键选中一个属性要求所有数据按该属性进行颜色插值
- 2. 点击右键恢复初始图表
- 3. 按住左键拖动，框选出对应属性范围内的数据，虚化不在范围内的数据
- 4. 实现多个属性范围的筛选

首先设置全局变量 first_draw 用于标识是否需要恢复到原图表 (false 标识需要恢复), 用于实现初始化绘制和右键恢复的操作。对于属性筛选，我们使用 mouse_mode 变量进行控制 Lclick 记为 mouse_mode=1, 用于实现功能 1(1. 点击左键选中一个属性要求所有数据按该属性进行颜色插值),Ldrag 记为 mouse_mode=2, 用于实现功能 3,4 (3. 按住左键拖动，框选出对应属性范围内的数据，虚化不在范围内的数据;4. 实现多个属性范围的筛选) 且 mouse_mode 无论为 1 还是 2，都需要考虑是否能够正确选中对应属性，select_bar 变量用于检测选中哪个属性，逻辑如下：

```
int select_bar = -1;
if (mouse_mode != 0)
{
    for (int i = 0; i < 7; i++)
    {
```

```
        if (abs(mouse_first_pos.x - coor_x[i]) < bar_width / 2)
        {
            mouse_first_pos.y  = std::max(bar_up, mouse_first_pos.y);
            mouse_first_pos.y  = std::min(bar_down, mouse_first_pos.y);
            mouse_second_pos.y = std::max(bar_up, mouse_second_pos.y);
            mouse_second_pos.y = std::min(bar_down, mouse_second_pos.y);

            select_bar = i;
            if (mouse_mode == 2)
            {
                select_bars[i]          = 1;
                select_bars_ypos[i][0] = mouse_first_pos.y;
                select_bars_ypos[i][1] = mouse_second_pos.y;
            }
            break;
        }
    }
}
```

监测鼠标活动：

```
int  mouse_mode      = 0;

glm::vec2 mouse_first_pos(0.0f, 0.0f);
glm::vec2 mouse_second_pos(0.0f, 0.0f);
if (proxy.IsHovering())
{
    if (proxy.IsClicking(true))
    {
        mouse_first_pos=proxy.MousePos();
        mouse_mode = 1;
    }
    else if (proxy.IsDragging(true))
    {
        mouse_mode = 2;
        mouse_first_pos = proxy.DraggingStartPoint();
        mouse_second_pos = proxy.MousePos();
    }
    else if (proxy.IsClicking(false))
    {
        first_draw=false;//refresh
        memset(select_bars, 0, sizeof(select_bars));
    }
}
```

然后对于不同情况进行绘制即可:

首先图表框架都是统一的：

```
static CoordinateStates states(data);
```

```
SetBackGround(input, glm::vec4(1));

// data process
float padding_percent = 0.1;
int max_vals[7];
int min_vals[7];
float   coor_x[7];
std::string properties_tag[7] = { "cylinders", "displacement", "weight", "horsepower", "
    acceleration(0-60mph)", "mileage", "year" };
for (int i = 0; i < 7; i++)
{
    int distance = int((states.max_values[i] - states.min_values[i])*padding_percent)+1;
    max_vals[i]  = states.max_values[i] + distance;
    min_vals[i]  = states.min_values[i] - distance;
    coor_x[i]    = (0.85 / 6)*i + 0.075;
}

// draw coordinates

float bar_up=0.15;
float bar_down = 0.9;
float bar_width = 0.04;
for (int i = 0; i < 7; i++)
{
    // Draw vertical line for each bar
    DrawLine(input, glm::vec4 { 0, 0, 0, 1 }, glm::vec2 { coor_x[i], bar_up }, glm::vec2 {
    coor_x[i], bar_down }, 3);
    DrawFilledRect(input, glm::vec4 { 0, 0.5, 0.5, 0.3 }, glm::vec2 { coor_x[i] - bar_width/2,
     bar_up }, glm::vec2 { bar_width, bar_down-bar_up });

    // Draw tag of each bar
    // properties tag
    PrintText(input, glm::vec4 { 0, 0, 0, 1 }, glm::vec2 { coor_x[i], 0.09 },0.015,
    properties_tag[i]);
    // max tag
    PrintText(input, glm::vec4 { 0, 0, 0, 1 }, glm::vec2 { coor_x[i], 0.125 },0.03, std::to_
    string(max_vals[i]));
    // min tag
    PrintText(input, glm::vec4 { 0, 0, 0, 1 }, glm::vec2 { coor_x[i], 0.925 },0.03, std::to_
    string(min_vals[i]));
}
```

初始化图表/恢复原图表：

```
if (first_draw == false) {
    int    data_num = states.data.size();
    float tmp_yi1, tmp_yi2;
    for (int i = 0; i < data_num; i++) {
        float tmp_data[7] = { (float) states.data[i].cylinders, states.data[i].displacement,
    states.data[i].weight, states.data[i].horsepower, states.data[i].acceleration, states.data[
    i].mileage, (float) states.data[i].year };
```

```
        tmp_yi1            = linear_interpolation(tmp_data[0], min_vals[0], max_vals[0], bar_down, bar_up);
        for (int j = 1; j < 7; j++) {
            tmp_yi2 = linear_interpolation(tmp_data[j], min_vals[j], max_vals[j], bar_down, bar_up);
            DrawLine(input, glm::vec4(1.0 - 0.9 * i / data_num, 0.2, 0.9 * i / data_num, origin_alpha), glm::vec2 { coor_x[j - 1], tmp_yi1 }, glm::vec2 { coor_x[j], tmp_yi2 }, 1.2);
            tmp_yi1 = tmp_yi2;
        }
    }
    first_draw = true;
    return true;
}
```

若 select_bar!=-1(表示能够正确选中属性)

若鼠标单击左键, 则实现要求所有数据按该属性进行颜色插值

```
if (mouse_mode == 1)
{
    DrawFilledRect(input, glm::vec4 { 1.0, 0.2, 0.1, 0.7 }, glm::vec2 { coor_x[select_bar] - bar_width / 2, bar_up }, glm::vec2 { bar_width, bar_down - bar_up });
    memset(select_bars, 0, sizeof(select_bars));
    int   data_num = states.data.size();
    float tmp_yi1, tmp_yi2;
    for (int i = 0; i < data_num; i++) {

        float tmp_data[7] = { (float) states.data[i].cylinders, states.data[i].displacement, states.data[i].weight, states.data[i].horsepower, states.data[i].acceleration, states.data[i].mileage, (float) states.data[i].year };
        float color_k     = (tmp_data[select_bar] - min_vals[select_bar]) / (max_vals[select_bar] - min_vals[select_bar]);

        tmp_yi1            = linear_interpolation(tmp_data[0], min_vals[0], max_vals[0], bar_down, bar_up);
        for (int j = 1; j < 7; j++) {
            tmp_yi2 = linear_interpolation(tmp_data[j], min_vals[j], max_vals[j], bar_down, bar_up);
            DrawLine(input, glm::vec4(0.9 * color_k, 0.2, 1.0-0.9 * color_k, origin_alpha), glm::vec2 { coor_x[j - 1], tmp_yi1 }, glm::vec2 { coor_x[j], tmp_yi2 }, 1.2);
            tmp_yi1 = tmp_yi2;
        }
    }
    return true;
}
```

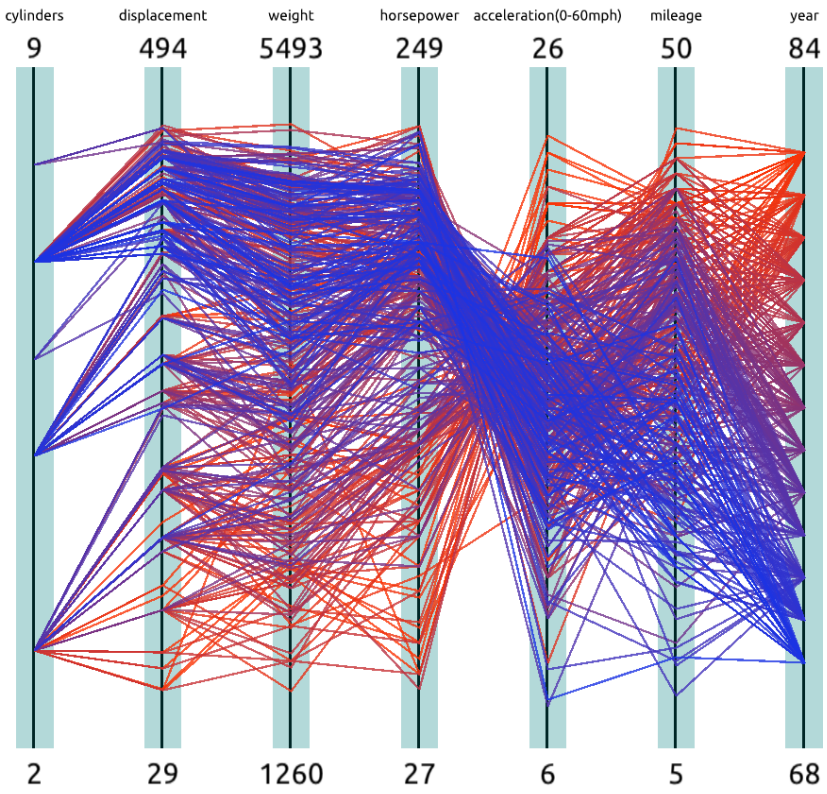
若按住鼠标左键然后拖动, 则实现筛选功能

```
else if (mouse_mode == 2)
```

```
{

    for (int i = 0; i < 7; i++)
    {
        if (select_bars[i] == 0) continue;
        DrawFilledRect(input, glm::vec4 { 1.0, 0.2, 0.1, 0.7 }, glm::vec2 { coor_x[i] - bar_width / 2, bar_up }, glm::vec2 { bar_width, bar_down - bar_up });
    }

    int   data_num = states.data.size();
    float tmp_yi1, tmp_yi2;
    for (int i = 0; i < data_num; i++) {
        float tmp_data[7] = { (float) states.data[i].cylinders, states.data[i].displacement, states.data[i].weight, states.data[i].horsepower, states.data[i].acceleration, states.data[i].mileage, (float) states.data[i].year };
        float color_k     = (tmp_data[select_bar] - min_vals[select_bar]) / (max_vals[select_bar] - min_vals[select_bar]);


        float tmp_alpha = origin_alpha;


        for (int j = 0; j < 7; j++)
        {
            if (select_bars[j] == 0) continue;
            float select_y = linear_interpolation(tmp_data[j], min_vals[j], max_vals[j], bar_down, bar_up);
            if (select_y < std::min(select_bars_ypos[j][0], select_bars_ypos[j][1]) || select_y > std::max(select_bars_ypos[j][0], select_bars_ypos[j][1]))
            {
                tmp_alpha = lower_alpha;
                break;
            }
        }


        tmp_yi1 = linear_interpolation(tmp_data[0], min_vals[0], max_vals[0], bar_down, bar_up);
        for (int j = 1; j < 7; j++) {
            tmp_yi2 = linear_interpolation(tmp_data[j], min_vals[j], max_vals[j], bar_down, bar_up);
            DrawLine(input, glm::vec4(0.9 * color_k, 0.2, 1.0 - 0.9 * color_k, tmp_alpha), glm::vec2 { coor_x[j - 1], tmp_yi1 }, glm::vec2 { coor_x[j], tmp_yi2 }, 1.2);
            tmp_yi1 = tmp_yi2;
        }
    }

    for (int j = 0; j < 7; j++) {
        if (select_bars[j] == 0) continue;
        DrawFilledRect(input, glm::vec4 { 1.0, 0.2, 0.1, 0.9 }, glm::vec2 { coor_x[j] - bar_width / 2, std::min(select_bars_ypos[j][0], select_bars_ypos[j][1]) }, glm::vec2 {
```

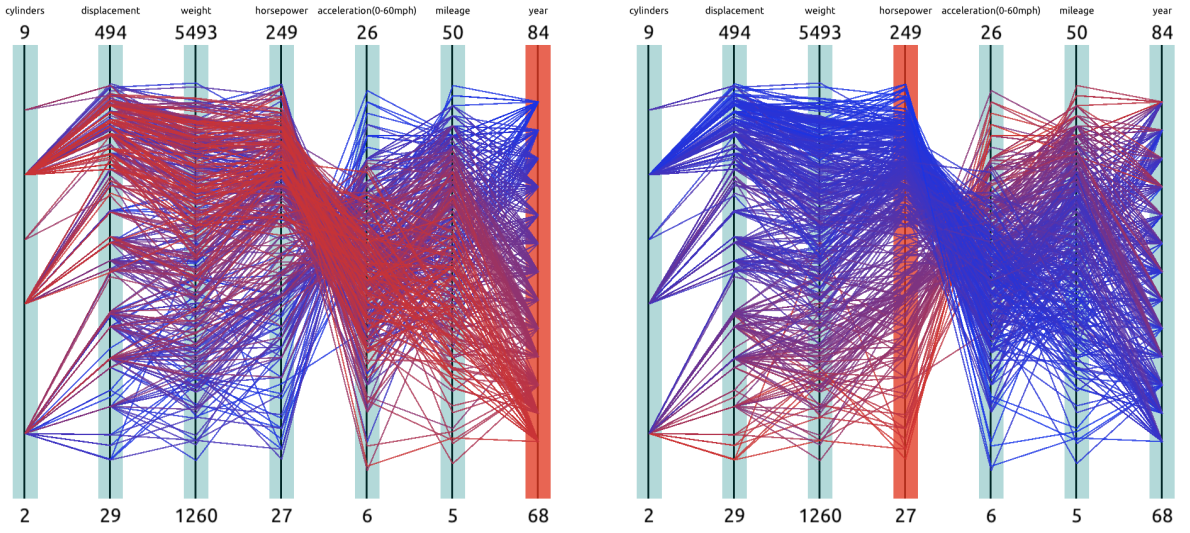
```
    bar_width, std::max(select_bars_ypos[j][0], select_bars_ypos[j][1]) - std::min(
    select_bars_ypos[j][0], select_bars_ypos[j][1]) });
43
44          float bar_height  = bar_down - bar_up;
45          int   val_distance = max_vals[j] - min_vals[j];
46          int   up_boundary   = max_vals[j]-(std::min(select_bars_ypos[j][0], select_bars_ypos[j
    ][1]) - bar_up) / bar_height * (max_vals[j] - min_vals[j]);
47          int   down_boundary = max_vals[j]-(std::max(select_bars_ypos[j][0], select_bars_ypos[j
    ][1]) - bar_up) / bar_height * (max_vals[j] - min_vals[j]);
48          // up boundary
49          PrintText(input, glm::vec4 { 0, 0, 0, 1 }, glm::vec2 { coor_x[j], std::min(
    select_bars_ypos[j][0], select_bars_ypos[j][1])-0.02 }, 0.02, std::to_string(up_boundary));
50          // down boundary
51          PrintText(input, glm::vec4 { 0, 0, 0, 1 }, glm::vec2 { coor_x[j], std::max(
    select_bars_ypos[j][0], select_bars_ypos[j][1])+0.02 }, 0.02, std::to_string(down_boundary)
    );
     }
53
54
     return true;
56 }
```

最终得到效果图如下:



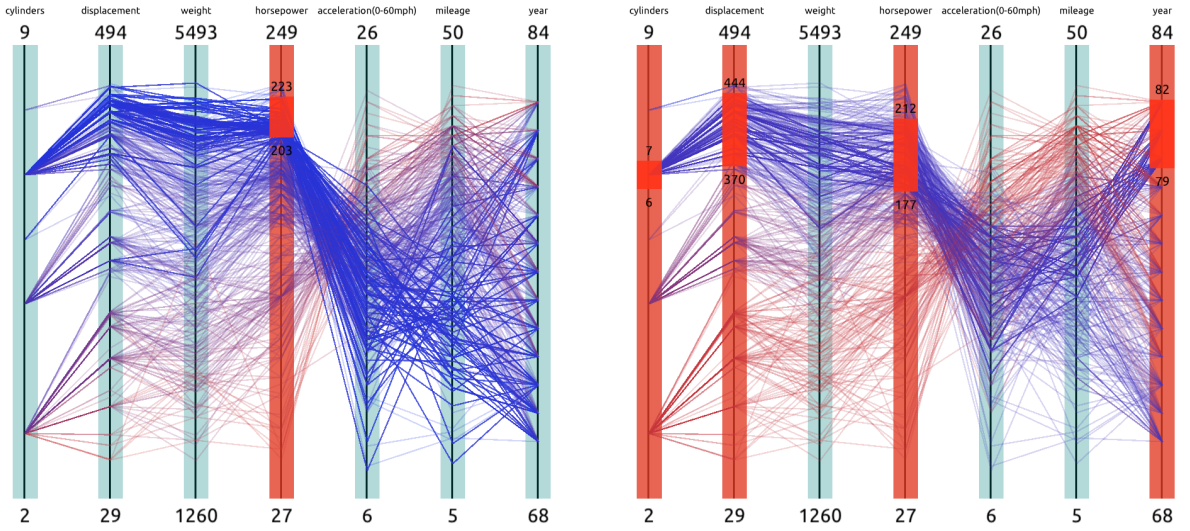
(a) Original Chart

图 1: Task 1-proxy functions



(a) Click to Select A property(year)　　(b) Click to Select A property(horsepower)

图 2: Task 1-proxy functions



(a) Drag to select a range of a property(horsepower)　　(b) Multy select ranges of properties

图 3: Task 1-proxy functions

2 Task2:Flow Visualization

整体思路：

按坐标依次遍历 2D 流场中的每个点，先正向积分 (沿对应向量正方向) 移动 step 长度，每次移动都更新 x 和 y 的坐标，利用随机噪音图像计算并累加正向积分和和正向权重和，然后同理进行反向积分 (沿对应向量反方向)，最后利用公式 $output(i,j) = \frac{forward\_sum(i,j)+backward\_sum(i,j)}{forward\_total(i,j)+backward\_total(i,j)}$ 计算得到对应点的可视化结果。

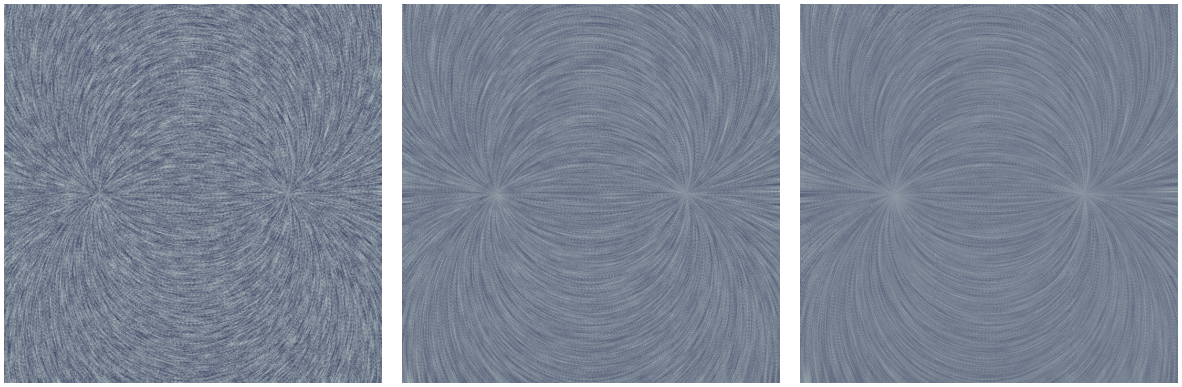
完整代码如下：

```
void LIC(ImageRGB & output, Common::ImageRGB const & noise, VectorField2D const & field, int
    const & step) {
    // your code here
```

```
    int width=noise.GetSizeX();
    int height=noise.GetSizeY();
    SetBackGround(output, glm::vec4(0));
    float t = 0;

    for (int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            // Forward
            float y =i;
            float x = j;
            glm::vec3 forward_sum   = {0.0f,0.0f,0.0f};
            float forward_total=0;

            for (int k = 0; k < step; k++)
            {
                float dx = field.At(int(x), int(y))[0];
                float dy = field.At(int(x), int(y))[1];
                float dt_x=0;
                float dt_y=0;
                float dt   = 0;

                if (dy > 0)dt_y = (floor(y)+1-y)/dy;
                else if (dy < 0) dt_y = -(y+1-ceil(y))/dy;

                if (dx > 0) dt_x = (floor(x) + 1 - x) / dx;
                else if (dx < 0) dt_x = -(x + 1 - ceil(x)) / dx;

                if (dx == 0 && dy == 0) dt = 0;
                else dt = std::min(dt_x, dt_y);

                x = std::min(std::max(x + dx * dt, float(0)), float(width - 1));
                y = std::min(std::max(y + dy * dt, float(0)), float(height - 1));

                float weight = pow(cos(t + 0.46 * (float) k), 2);
                forward_sum += noise.At(int(x), int(y)) * weight;
                forward_total += weight;

            }

            // Backward
            y  = i;
            x  = j;
            glm::vec3 backward_sum   = { 0.0f, 0.0f,0.0f};
            float     backward_total = 0;

            for (int k = 1; k < step; k++) {
                float dx   = -field.At(int(x), int(y))[0];
                float dy   = -field.At(int(x), int(y))[1];
                float dt_x = 0;
```

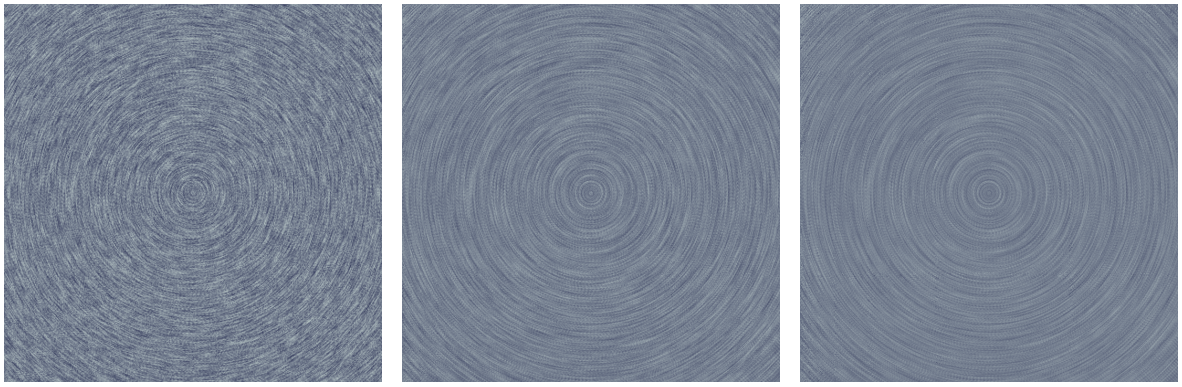
```
                float dt_y = 0;
                float dt   = 0;

                if (dy > 0) dt_y = (floor(y) + 1 - y) / dy;
                else if (dy < 0) dt_y = -(y + 1 - ceil(y)) / dy;

                if (dx > 0) dt_x = (floor(x) + 1 - x) / dx;
                else if (dx < 0) dt_x = -(x + 1 - ceil(x)) / dx;

                if (dx == 0 && dy == 0) dt = 0;
                else dt = std::min(dt_x, dt_y);

                x = std::min(std::max(x + dx * dt, float(0)), float(width - 1));
                y = std::min(std::max(y + dy * dt, float(0)), float(height - 1));

                float weight = pow(cos(t + 0.46 * (float)k), 2);
                backward_sum += noise.At(int(x), int(y)) * weight;
                backward_total += weight;
            }

            output.At(j,i) = (forward_sum + backward_sum) / (forward_total + backward_total);
        }
    }
}
```

得到效果图如下：



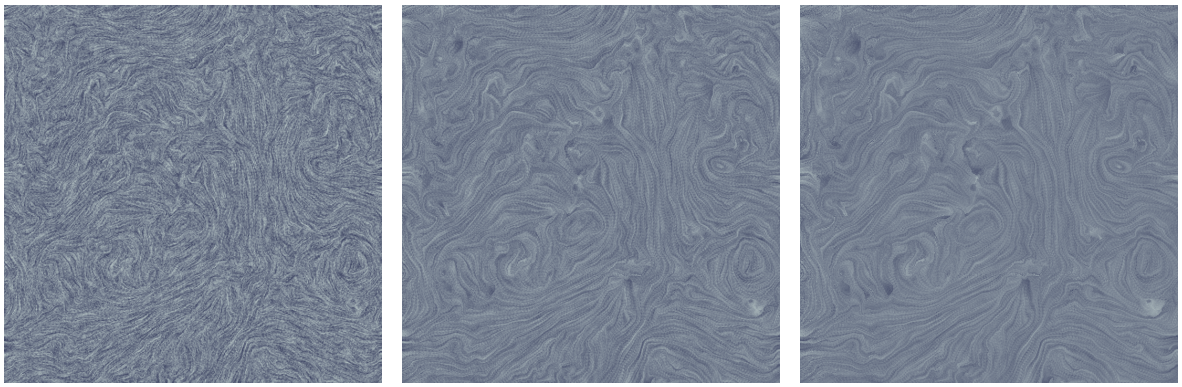
(a) LIC:Bipole step=10　(b) LIC:Bipole step-30　(c) LIC:Bipole step=50

图 4: Task 2:LIC-Bipole



(a) LIC:Circle step=10 (b) LIC:Circle step=30 (c) LIC:Circle step=50

图 5: Task 2:LIC-Circle



(a) LIC:Turbulence step=10 (b) LIC:Turbulence step=30 (c) LIC:Turbulence step=50

图 6: Task 2:LIC-Turbulence