HW3 Report

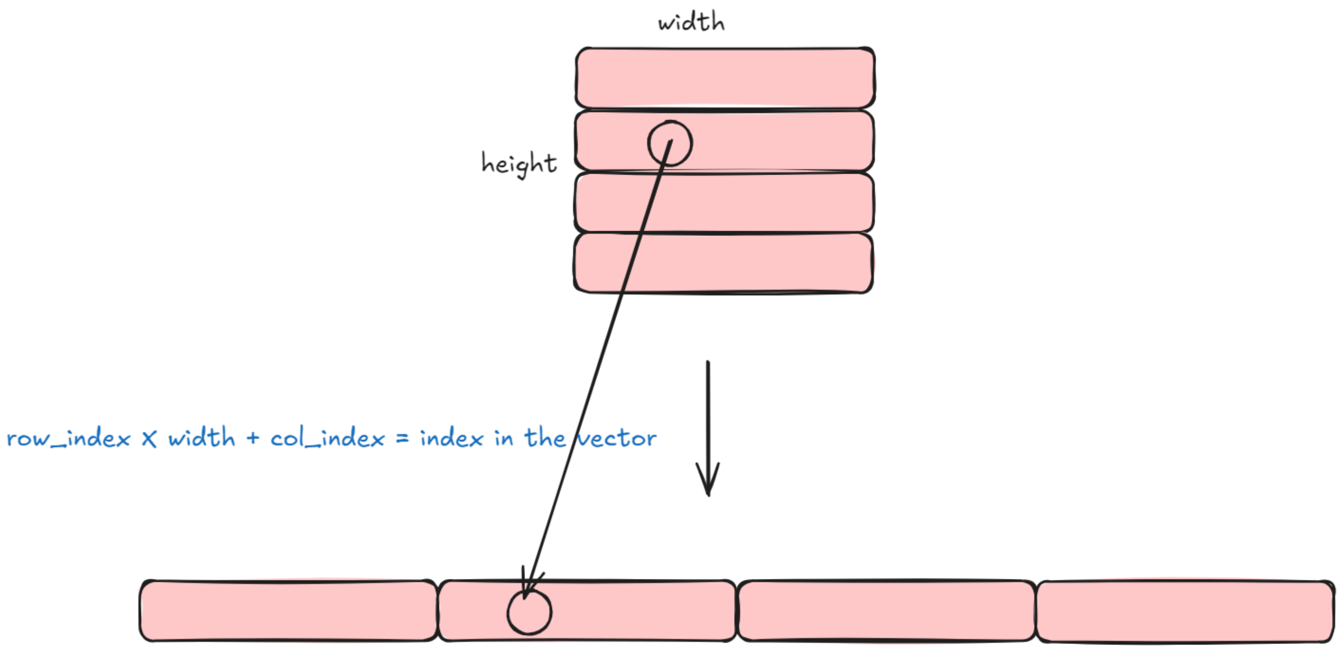
HW3 要实现的效果非常直观，代码流程也不复杂，最关键最难的地方在于如何实现 `build_poisson_equation`

首先我们要明确一些概念什么是 `src`，`tar`，`mask`

- `src` and `mask` come from the same picture, e.g. `bear`
- `tar` is the result picture, e.g. `sea`

index_map

由于涉及到把图像拉成一维向量，这个映射可以随便定，但重点是我们需要知道这个 `map`



我们有下面的代码

```
index_map.clear();
int index = 0;
for (int y = 0; y < height; ++y)
{
    for (int x = 0; x < width; ++x)
    {
        if (mask->get_pixel(x, y)[0] > 128)
        {
            index_map[y * width + x] = index++;
        }
    }
}
const int N = index_map.size();
```

How to build the poisson equation

让我们先来回顾一下原论文的说法

> As it is enough to solve the interpolation problem for each color component separately, we consider only scalar image functions.

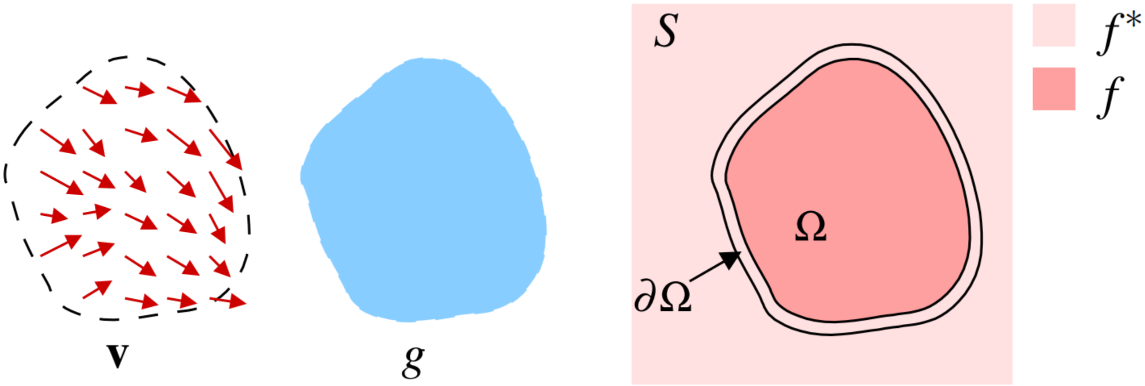


Figure 1: **Guided interpolation notations**. Unknown function f interpolates in domain Ω the destination function f^*, under guidance of vector field $\mathbf{v}$, which might be or not the gradient field of a source function g.

本质上是一个插值的工作，$f^\star$ is the known scalar function defined over S minus the interior of Ω, f is the function we wanna to know, i.e. the unknown scalar function defined over the interior of Ω. Of course, we should satisfy the boundary condition. Except that, we know the intepolation should follow some guidance, the guidance we use here is the **vector field v** defined over Ω

Minimal equation

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \quad \text{with } f|_{\partial\Omega} = f^\star|_{\partial\Omega}$$

The minimizer above is the same as the solution of the Poisson equation with Dirichlet boundary conditions:

$$\Delta f = \operatorname{div}\mathbf{v} \text{ over } \Omega, \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega},$$

if $\mathbf{v}$ is conservative, i.e. it's the gradient of some function g, then we can consider the

$$\Delta \tilde{f} = 0 \text{ over } \Omega, \ \tilde{f}|_{\partial\Omega} = (f^* - g)|_{\partial\Omega}.$$

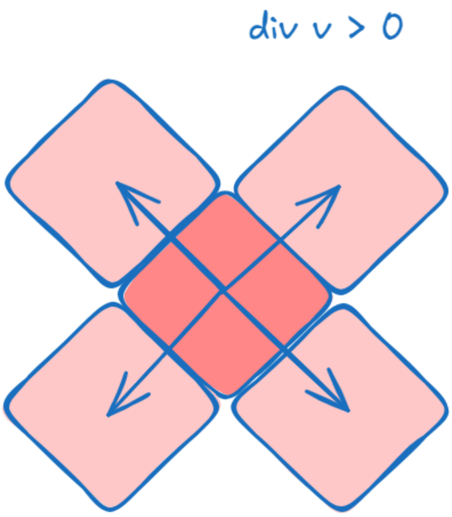
Discrete Poisson solver

- For each pixel p in S, let N_p be the set of its 4-connected neighbors which are in S.
- $\langle p, q\rangle$ denote a pixel pair such that $q \in N_p$
- f_p denote the value of f at p
- The boundary of Ω is now $\partial\Omega = \{p \in S \setminus \Omega : N_p \cap \Omega \neq \emptyset\}$.

Discrete Laplacian

Δf(i, j) ≈ f(i+1, j) + f(i-1, j) + f(i, j+1) + f(i, j-1) - 4f(i, j)

Discrete Divergence of the Vector Field



Just like the *flow*

But the picture above is Wrong!

We should remember the divergence of a vector field means the *outgoingness*

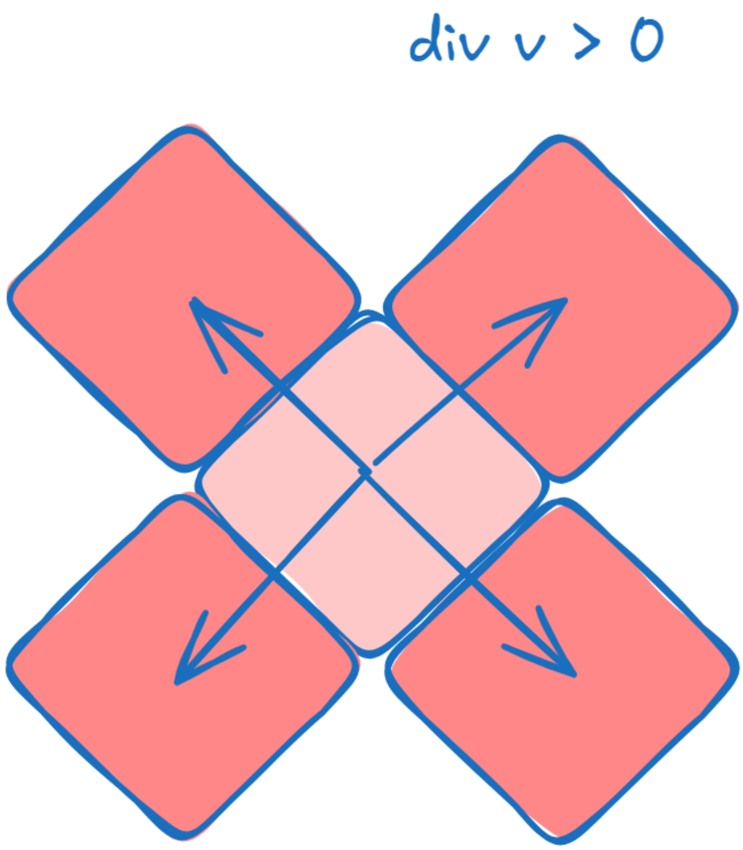
That is the *outflow* - *inflow*

We should consider each direction individually,

_outflow_x - inflow_x = g(x+1, y) - g(x,y) - (g(x,y) - g(x-1, y))

Then we have _outflow - inflow = g(x+1,y)+g(x,y+1)+g(x-1,y)+g(x,y-1) - 4g(x,y)

Then $-\nabla \cdot \nabla g_p = \sum_{q \in N_p} g_p - g_q$



有点违反直觉

Now we get the Discrete Poisson Equation

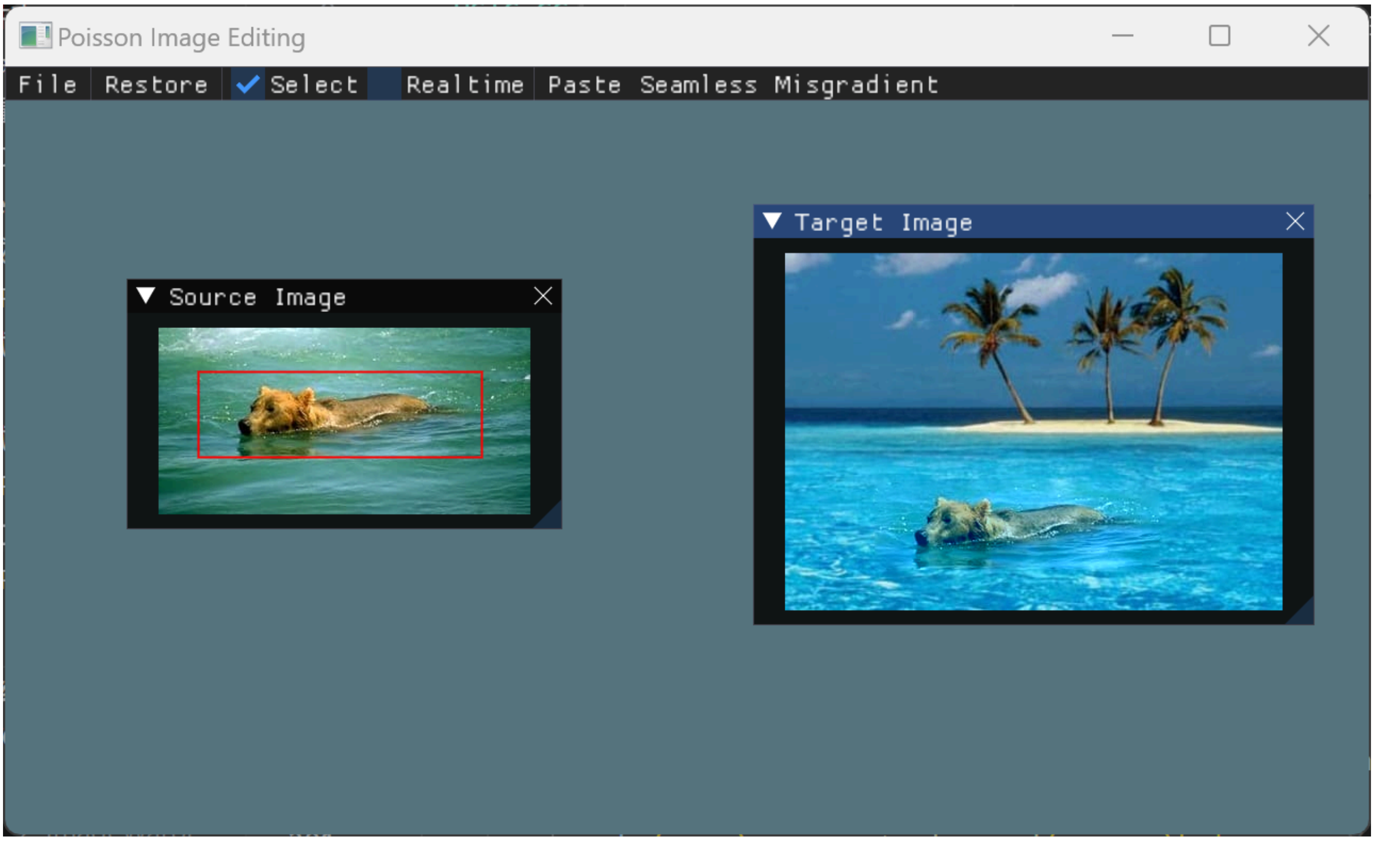
$$\text{for all } p \in \Omega, \quad |N_p| f_p - \sum_{q \in N_p \cap \Omega} f_q = \sum_{q \in N_p \cap \partial\Omega} f_q^* + \sum_{q \in N_p} v_{pq}.$$

$v_{pq} = g_p - g_q$

$\sum_{q \in N_p} v_{pq} = |N_p| g_p - \sum_{q \in N_p} g_q$

Turn to the Code

We succeed!



有了上面的铺垫，加上Deepseek/Gemini/Claude，写出正确的代码是自然的事情😄

Now Let's consider `Mixing gradients`

The discrete counterpart of this guidance field is:

$$v_{pq} = \begin{cases} f_p^* - f_q^* & \text{if } |f_p^* - f_q^*| > |g_p - g_q|, \\ g_p - g_q & \text{otherwise,} \end{cases} \tag{13}$$

for all $\langle p, q \rangle$. The effect of this guidance field is demonstrated in

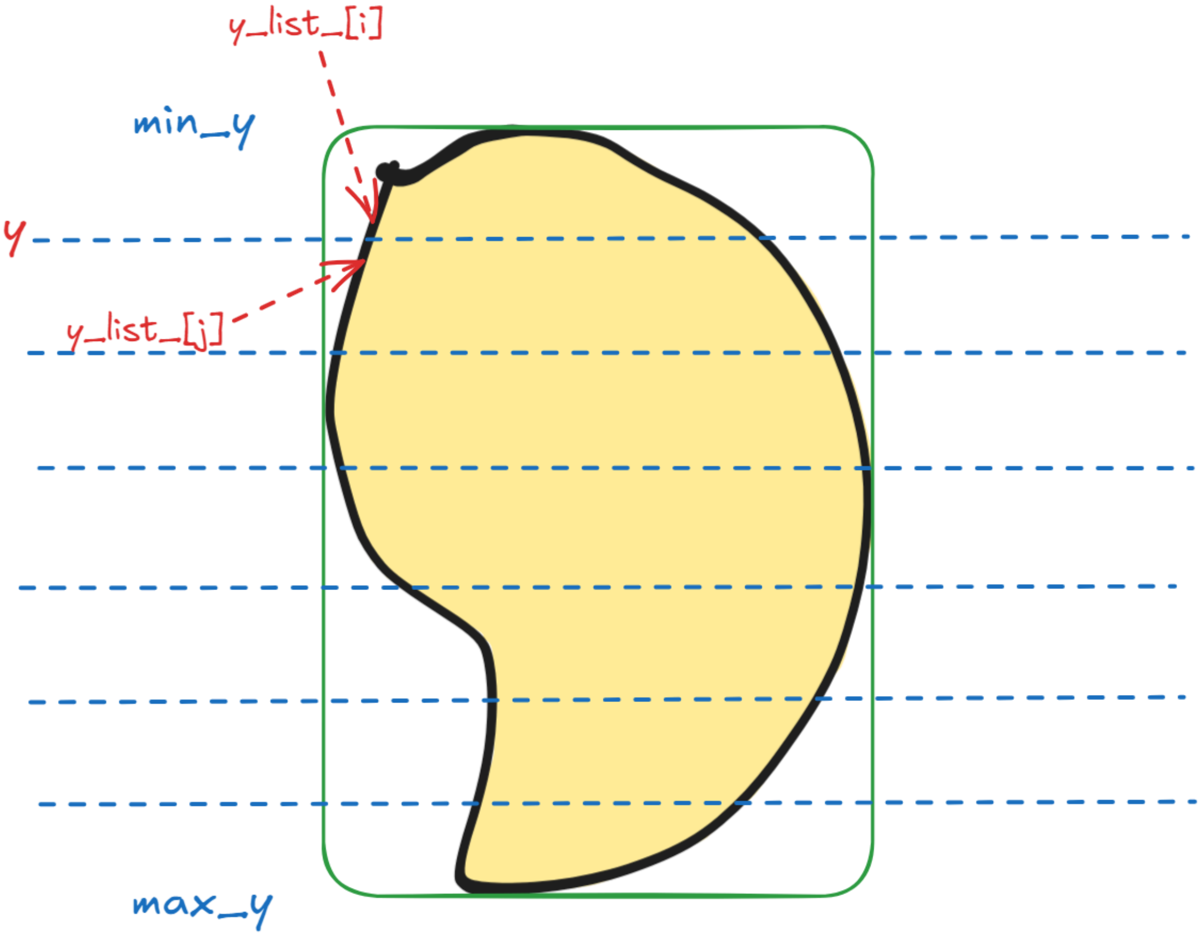
```cpp
for (int c = 0; c < 3; ++c)
{
    const double src_val = src->get_pixel(x, y)[c];
    const double tar_val =
        tar->get_pixel(x + get_offset_x(), y + get_offset_y())[c];
    for (const auto& [nx, ny] : neighbors)
    {
        if (nx >= 0 && nx < width && ny >= 0 && ny < height)
        {
            if (std::abs(
                    tar_val -
                    tar->get_pixel(
                        nx + get_offset_x(), ny + get_offset_y())[c]) >
                std::abs(src_val - src->get_pixel(nx, ny)[c]))
            {
                b_(i, c) += tar_val - tar->get_pixel(
                                            nx + get_offset_x(),
                                            ny + get_offset_y())[c];
            }
            else
            {
                b_(i, c) += src_val - src->get_pixel(nx, ny)[c];
            }
        }
    }
}
```

Mixing gradient is easy to implement on the base code

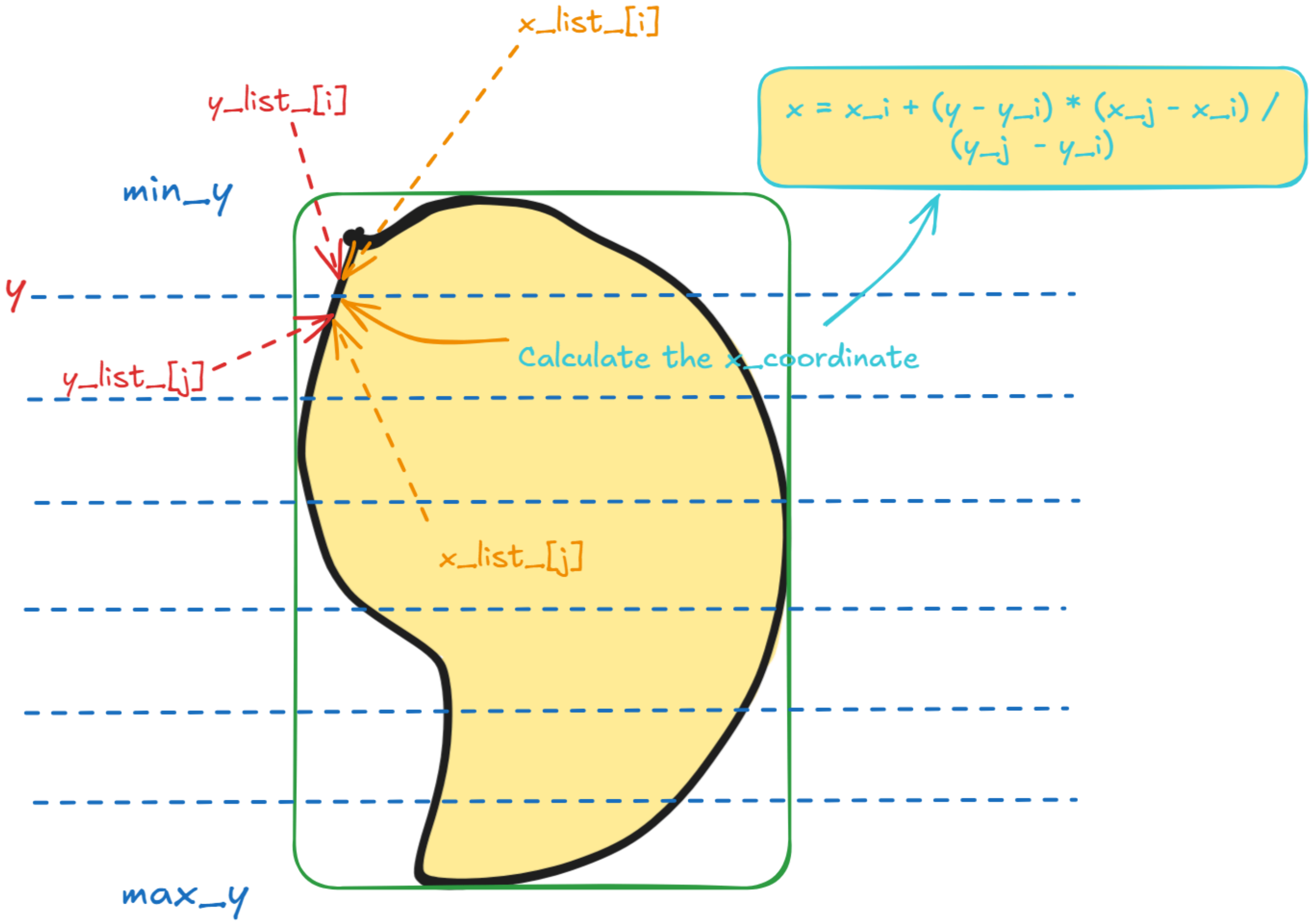
Now let's consider `Freehand` selected_region

`Freehand` 最重要的实现就是如何得到内部点 `get_interior_pixels`

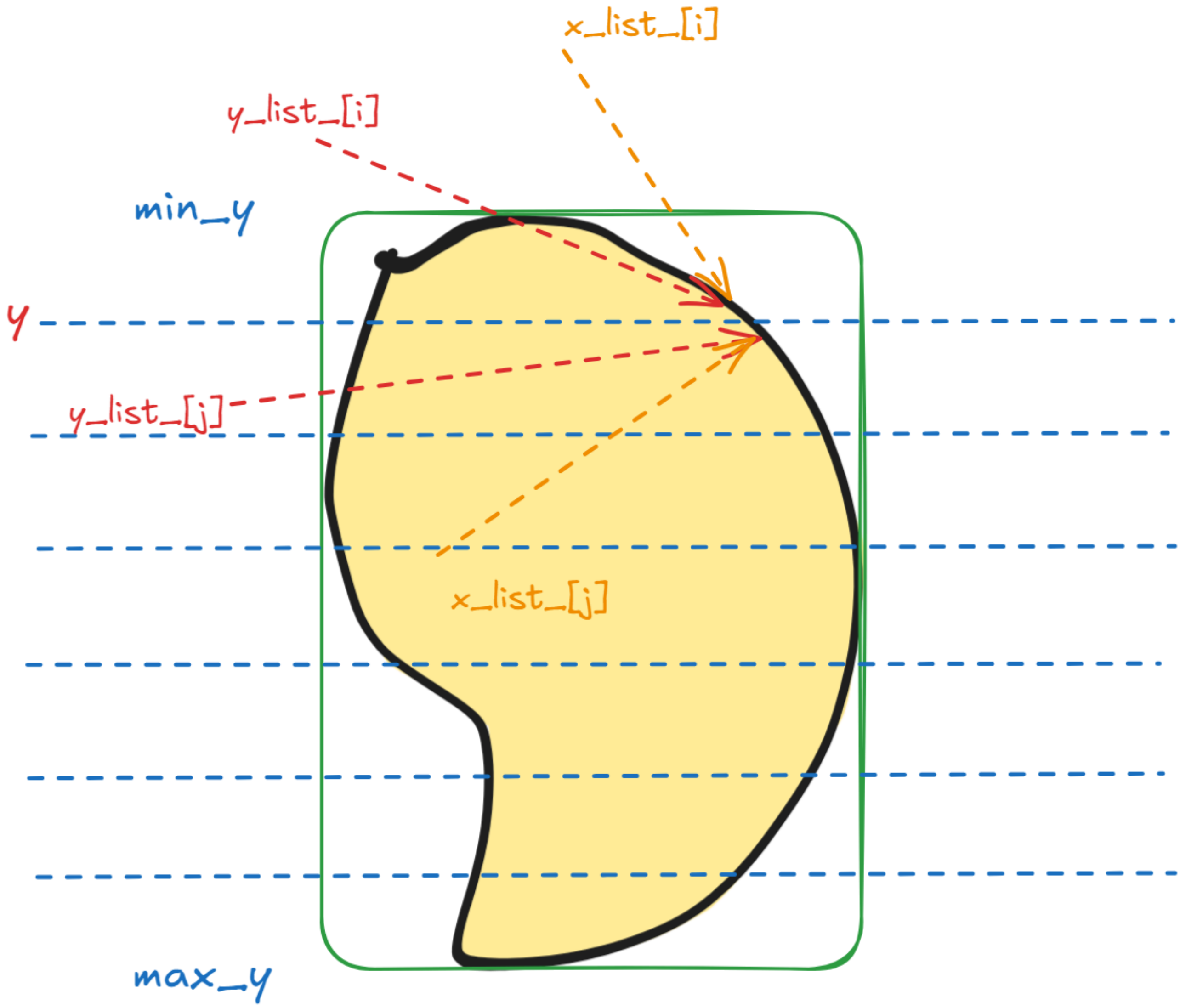
我们看图就知道算法是如何写的



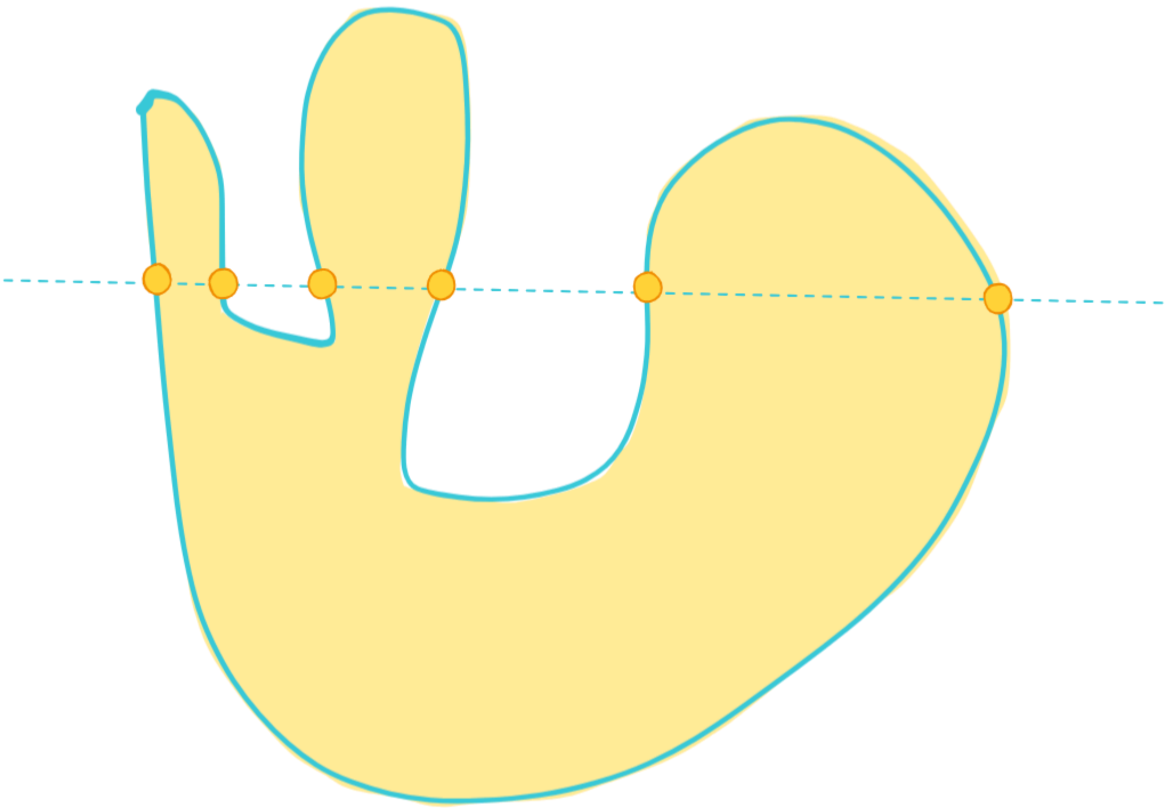
找到左右交点`intersections`的横坐标`x_left and x_right`，就可以把所有中间点存到`interior_pixels`当中



对于右边的交点也是一样



现在只差最后一步了，这一步非常关键，我们要考虑交点不止有两个情况



于是我们需要

```
    // Sort intersections
    std::sort(intersections.begin(), intersections.end());

    // Fill pixels between pairs of intersections
    for (size_t i = 0; i < intersections.size(); i += 2) {
        if (i + 1 >= intersections.size()) break;
        for (int x = intersections[i]; x <= intersections[i + 1]; ++x) {
            interior_pixels.emplace_back(x, y);
        }
    }
}
```

Tips

此次作业增加了`logger`打印日志功能，虽然写代码的时候要多些`logger`还挺麻烦的，但是`logger`打印出来各个通道求解的`pixel`的最小值和最大值，对于debug起到了关键的作用

结果展示

链接：谌奕同_Poisson_editing_ustc_cg
密码：2jy6