

HW2: report

姓名：杜朋澈

ID: 68

学号：PB21050988

HW2: report

算法

IDW warping

原理

实现

RBF warping

原理

实现

ANN inpainting

原理

实现

演示

IDW方法

RBF方法

总结

算法

图像变形是对数字图像进行几何变形的过程。给定一组分散平移向量的图像扭曲问题本质上是同时插值两个函数的问题。一个函数用于x轴方向的平移，另一个用于y轴方向的平移。形式化的来说：

给定 n 对控制点 $(\mathbf{p}_i, \mathbf{q}_i)$ ，其中 $\mathbf{p}_i, \mathbf{q}_i \in \mathbb{R}^2$ ， $i = 1, 2, \dots, n$ ，

希望得到一个函数 $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ ，满足插值条件：

$$f(\mathbf{p}_i) = \mathbf{q}_i, \quad \text{for } i = 1, 2, \dots, n.$$

IDW warping

原理

假设所求的插值函数 f 具有如下加权平均的形式

$$f(\mathbf{p}) = \sum_{i=1}^n w_i(\mathbf{p}) f_i(\mathbf{p}),$$

其中每个 f_i 是点 \mathbf{p}_i 处的局部近似 (local approximation)，满足在 $(\mathbf{p}_i, \mathbf{q}_i)$ 处的插值性质；权重函数 w_i 满足非负性和归一性

$$\sum_{i=1}^n w_i = 1, \text{ and } w_i \geq 0, \text{ for } i = 1, 2, \dots, n,$$

且在 \mathbf{p}_i 处 $w_i(\mathbf{p}_i) = 1$.

对于任意的 i ,

- 权重 $w_i : \mathbb{R}^2 \rightarrow \mathbb{R}$ 形如

$$w_i(\mathbf{p}) = \frac{\sigma_i(\mathbf{p})}{\sum_{j=1}^n \sigma_j(\mathbf{p})},$$

$$\text{可选 } \sigma_i(\mathbf{p}) = \frac{1}{\|\mathbf{p} - \mathbf{p}_i\|^\mu}, \quad \mu > 1.$$

- 映射 $f_i : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ 形如

$$f_i(\mathbf{p}) = \mathbf{q}_i + \mathbf{D}_i(\mathbf{p} - \mathbf{p}_i),$$

其中 $\mathbf{D}_i : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, 满足 $\mathbf{D}_i(\mathbf{0}) = \mathbf{0}$. 可选 \mathbf{D}_i 为线性变换, 即 $\mathbf{D}_i \in \mathbb{R}^{2 \times 2}$.

这里我们选取 $\mathbf{D}_i = \mathbf{I}$ 为恒等变换。

$$f(\mathbf{p}) = \sum_{i=1}^n w_i(\mathbf{p})(\mathbf{q}_i + \mathbf{p} - \mathbf{p}_i).$$

实现

将公式变形为:

$$f(\mathbf{p}) = \sum_{i=1}^n \frac{\sigma_i(\mathbf{p})}{\sum_{j=1}^n \sigma_j(\mathbf{p})} (\mathbf{q}_i + \mathbf{p} - \mathbf{p}_i) = \mathbf{p} + \sum_{i=1}^n \frac{\sigma_i(\mathbf{p})}{\sum_{j=1}^n \sigma_j(\mathbf{p})} (\mathbf{q}_i - \mathbf{p}_i).$$

其中:

$$\sigma_i(\mathbf{p}) = \frac{1}{\|\mathbf{p} - \mathbf{p}_i\|^\mu}$$

```

1  inline float sigma(int x, int y, float u, ImVec2 p_i) const
2  {
3      float norm =
4          std::sqrt((p_i.x - x) * (p_i.x - x) + (p_i.y - y) * (p_i.y
- y));
5      float frac = std::pow(norm, u);
6      return 1 / frac;
7  }
```

则插值函数 $f(p)$ 对应 `std::pair<int, int> WarperIDW::warping(int x, int y) const` 实现为:

```

1  std::pair<int, int> WarperIDW::warping(int x, int y) const
2  {
```

```

3     std::pair<float, float> sum(0, 0);
4     float deno = 0;
5     for (int j = 0; j < n; j++)
6     {
7         deno += sigma(x, y, mu, starts[j]);
8     }
9     for (int i = 0; i < n; i++)
10    {
11        float w = sigma(x, y, mu, starts[i]) / deno;
12        sum.first += w * (ends[i].x - starts[i].x);
13        sum.second += w * (ends[i].y - starts[i].y);
14    }
15    return std::pair<int, int>(
16        static_cast<int>(sum.first + x), static_cast<int>(sum.second +
17        y));
18 }
```

RBF warping

原理

假设所求的插值函数 f 是如下径向基函数组合的形式

$$f(\mathbf{p}) = \sum_{i=1}^n \alpha_i R(\|\mathbf{p} - \mathbf{p}_i\|) + A\mathbf{p} + \mathbf{b},$$

其中,

- $R(\|\mathbf{p} - \mathbf{p}_i\|)$ 是 n 个径向基函数, 其系数 $\alpha_i \in \mathbb{R}^2$ 待定;
- $A \in \mathbb{R}^{2 \times 2}$ 和 $\mathbf{b} \in \mathbb{R}^2$ 是待定的仿射部分.

此处选择径向基函数为 $R(d) = (d^2 + r^2)^{1/2}$

该映射有 $2(n + 3)$ 个自由度, 插值条件

$$f(\mathbf{p}_j) = \sum_{i=1}^n \alpha_i R(\|\mathbf{p}_j - \mathbf{p}_i\|) + A\mathbf{p}_j + \mathbf{b} = \mathbf{q}_j, \quad j = 1, \dots, n.$$

提供了 $2n$ 个约束, 补充约束为

$$\begin{pmatrix} \mathbf{p}_1 & \cdots & \mathbf{p}_n \\ 1 & \cdots & 1 \end{pmatrix}_{3 \times n} \begin{pmatrix} \alpha_1^\top \\ \vdots \\ \alpha_n^\top \end{pmatrix}_{n \times 2} = \mathbf{0}_{3 \times 2}.$$

实现

$$\text{待定系数和仿射矩阵 } \mathbf{coef}_{(n+3) \times 2} = \begin{pmatrix} \boldsymbol{\alpha}_1^\top \\ \vdots \\ \boldsymbol{\alpha}_n^\top \\ A_1^\top \\ A_2^\top \\ b^\top \end{pmatrix}_{(n+3) \times 2}$$

可通过求解如下线性方程组获得：

$$\begin{pmatrix} R(\|\mathbf{p}_1 - \mathbf{p}_1\|) & \cdots & R(\|\mathbf{p}_1 - \mathbf{p}_n\|) & \mathbf{p}_1^\top & 1 \\ \vdots & & \vdots & \vdots & \vdots \\ R(\|\mathbf{p}_n - \mathbf{p}_1\|) & \cdots & R(\|\mathbf{p}_n - \mathbf{p}_n\|) & \mathbf{p}_n^\top & 1 \\ \mathbf{p}_1 & \cdots & \mathbf{p}_n & & \\ 1 & \cdots & 1 & 0 & \end{pmatrix}_{(n+3) \times (n+3)} \quad \mathbf{coef}_{(n+3) \times 2} = \begin{pmatrix} \mathbf{q}_1^\top \\ \vdots \\ \mathbf{q}_n^\top \\ 0 \end{pmatrix}_{(n+3) \times 2}$$

该操作通过Eigen库完成即可，具体实现如下（已提供必要注释）：

```
1 // 系数矩阵A
2 Eigen::SparseMatrix<float> A(n + 3, n + 3);
3
4 // 填充系数矩阵左上角 n x n 处
5 for (int i = 0; i < n; i++)
6 {
7     for (int j = 0; j < n; j++)
8     {
9         float r = euclidean_distance(starts[i], starts[j]);
10        float value = radial_basis_function(r, sigma);
11        A.insert(i, j) = value;
12    }
13 }
14 // 补充约束填充
15 for (int i = 0; i < n; i++)
16 {
17     A.insert(i, n) = starts[i].x;
18     A.insert(i, n + 1) = starts[i].y;
19     A.insert(i, n + 2) = 1;
20 }
21 for (int j = 0; j < n; j++)
22 {
23     A.insert(n, j) = starts[j].x;
24     A.insert(n + 1, j) = starts[j].y;
25     A.insert(n + 2, j) = 1;
26 }
27 // 稀疏矩阵会将剩余部分自动置零
28 A.makeCompressed();
29
30 // 常数矩阵b
```

```

31     Eigen::SparseMatrix<float> b(n + 3, 2);
32
33     for (int i = 0; i < n; i++)
34     {
35         b.insert(i, 0) = ends[i].x;
36         b.insert(i, 1) = ends[i].y;
37     }
38     // 稀疏矩阵会将剩余部分自动置零
39     b.makeCompressed();
40
41     // 使用LU分解求解
42     Eigen::SparseLU<Eigen::SparseMatrix<float>> solver;
43
44     solver.compute(A);
45
46     if (solver.info() != Eigen::Success)
47     {
48         std::cerr << "LU decomposition failed." << std::endl;
49         return;
50     }
51
52     coef = solver.solve(b);
53
54     // 检查求解是否成功
55     if (solver.info() != Eigen::Success)
56     {
57         std::cerr << "Linear system solving failed." << std::endl;
58         return;
59     }

```

获得待定系数和仿射矩阵 `coef` 后，即可代入进行插值函数的计算：

```

1 std::pair<int, int> WarperRBF::warping(int x, int y) const
2 {
3     float x_new = 0;
4     float y_new = 0;
5     for (int i = 0; i < n; i++)
6     {
7         float r = euclidean_distance(ImVec2(x, y), starts[i]);
8         float value = radial_basis_function(r, sigma);
9         x_new += coef.coeff(i, 0) * value;
10        y_new += coef.coeff(i, 1) * value;
11    }
12    x_new +=
13        coef.coeff(n, 0) * x + coef.coeff(n + 1, 0) * y + coef.coeff(n
14 + 2, 0);
15    y_new +=
16        coef.coeff(n, 1) * x + coef.coeff(n + 1, 1) * y + coef.coeff(n
17 + 2, 1);
18
19    return std::pair<int, int>(x_new, y_new);

```

ANN inpainting

原理

通过上述算法计算的插值函数通常既非满射也非单射，所以在每次计算出原像对应的像后，将该像加入到 `AnnoyIndex` 中建立索引，随后再次遍历处理后的图像，找到没有原像的元素点，在 `AnnoyIndex` 中对该像素点求取若干最近邻，获得它们的平均颜色作为填补输出。

实现

在图像变换过程中对构建索引：

```

1  Annoy::AnnoyIndex<
2      int,
3      float,
4      Annoy::Euclidean,
5      Annoy::Kiss32Random,
6      Annoy::AnnoyIndexSingleThreadedBuildPolicy>
7      index(2);
8  float i = 0;
9
10 float height = data_->height();
11 float width = data_->width();
12 for (int y = 0; y < height; ++y)
13 {
14     for (int x = 0; x < width; ++x)
15     {
16         //执行图像变换算法
17         auto [new_x, new_y] = warper_->warping(x, y);
18         //对于符合条件的像
19         if (new_x >= 0 && new_x < width && new_y >= 0 && new_y <
height)
20         {
21             std::vector<unsigned char> pixel = data_->get_pixel(x,
y);
22             warped_image.set_pixel(new_x, new_y, pixel);
23
24             //映射到-1到1的空间中，加入到索引
25             const float vec[2] = { (2 * new_x - width) / width,
26                                   (2 * new_y - height) / height
27             };
28             index.add_item(i, vec);
29             i++;
30         }
31     }

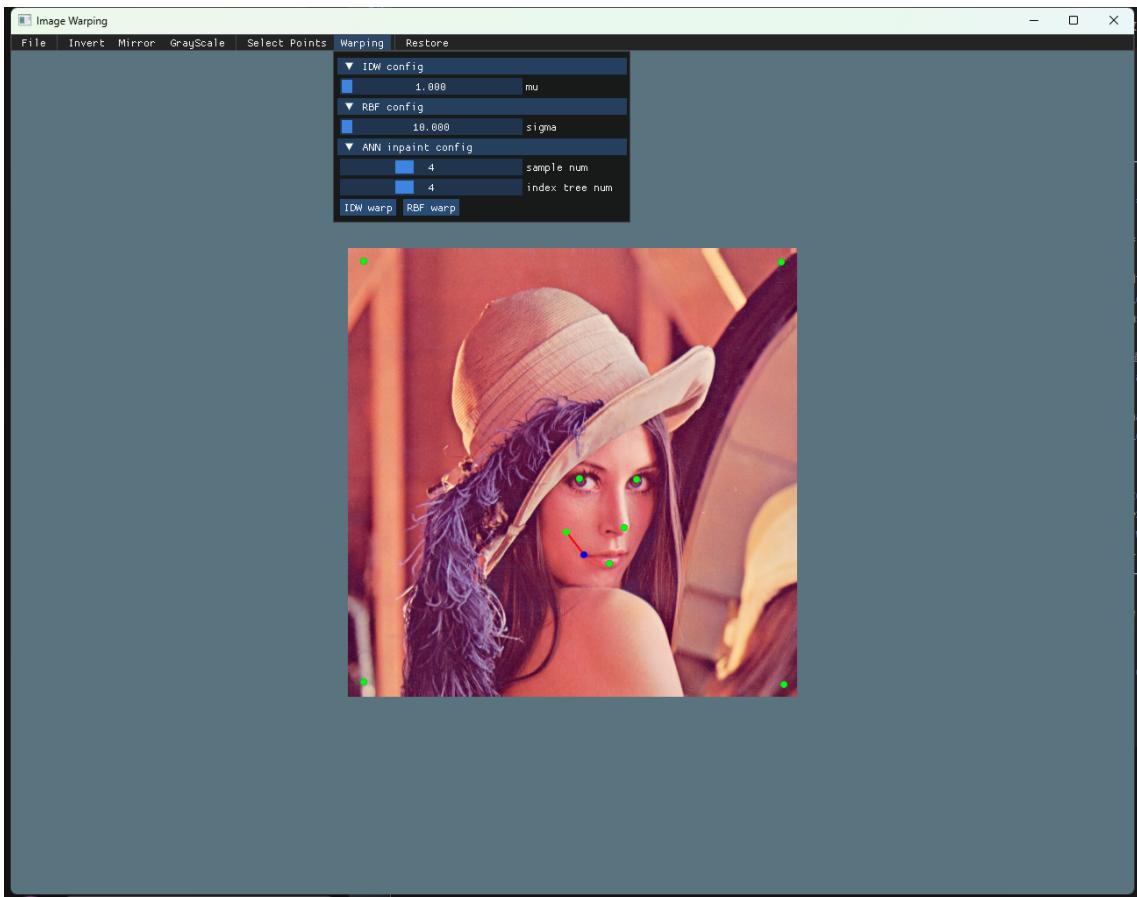
```

对处理后的图像运行ANN补洞算法，具体来说，我们提供两个配置参数 `ann_sample_num` 和 `ann_index_tree_num` 用于执行该算法：

```
1 //ANN 补洞
2 index.build(ann_index_tree_num);
3
4 for (int y = 0; y < data_->height(); ++y)
5 {
6     for (int x = 0; x < data_->width(); ++x)
7     {
8         std::vector<unsigned char> pixel =
9             warped_image.get_pixel(x, y);
10        //将纯黑色视为没有原像的像素点
11        if (pixel[0] == 0 and pixel[1] == 0 and pixel[2] == 0)
12        {
13            float vec[2] = { (2 * x - width) / width,
14                            (2 * y - height) / height };
15            std::vector<int> closest_items;
16            std::vector<float> distances;
17            //config: 求取ann_sample_num个最近邻像
18            index.get_nns_by_vector(vec, ann_sample_num, -1,
19            &closest_items, &distances);
20            //获取他们的平均颜色作为此处的输出
21            std::vector<unsigned char> channels(3, 0);
22            for (int j = 0; j < ann_sample_num; j++)
23            {
24                float result[2];
25                index.get_item(closest_items[j], result);
26                std::vector<unsigned char> sample =
27                    warped_image.get_pixel(
28                        (result[0] * width + width) / 2,
29                        (result[1] * height + height) / 2);
30                for (int i = 0; i < 3; i++)
31                {
32                    channels[i] += sample[i] / ann_sample_num;
33                }
34            }
35        }
```

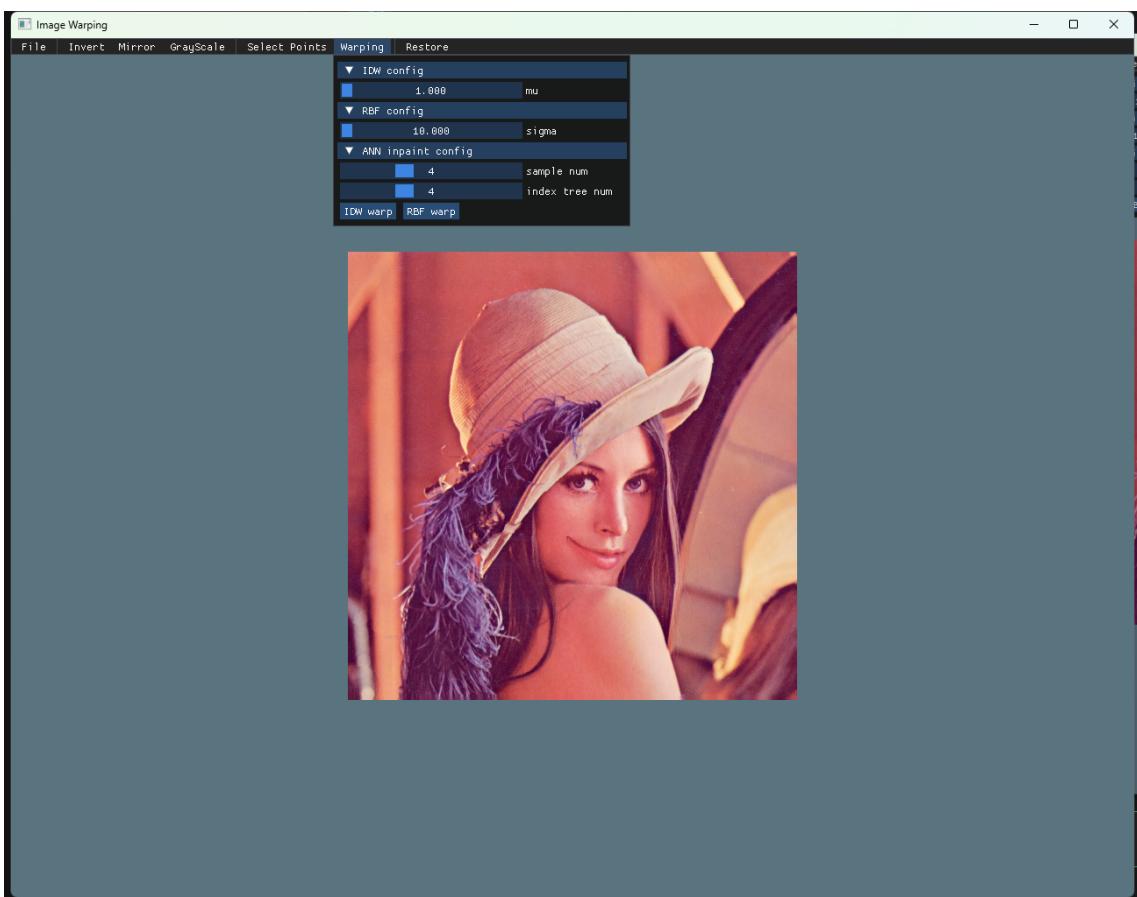
演示

- 选取控制点：

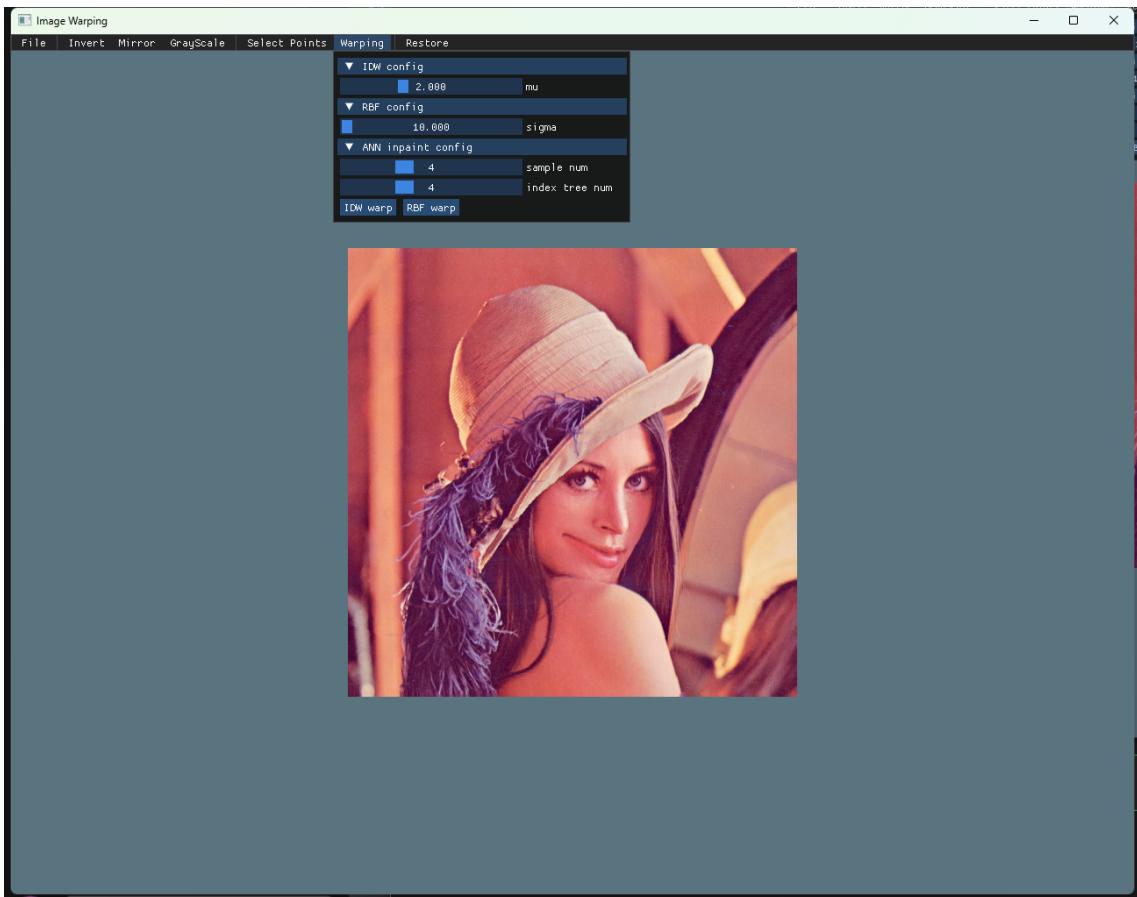


IDW方法

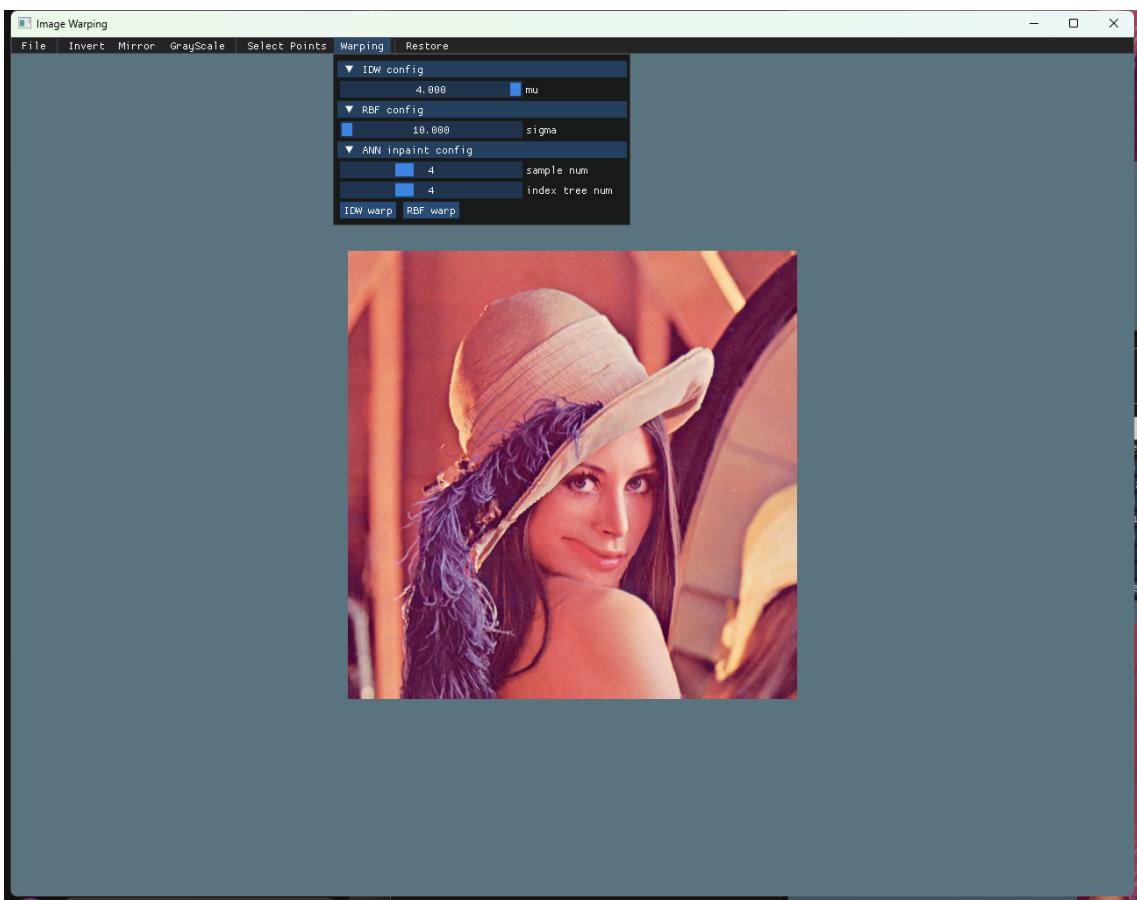
- $\mu = 1$



- $\mu = 2$



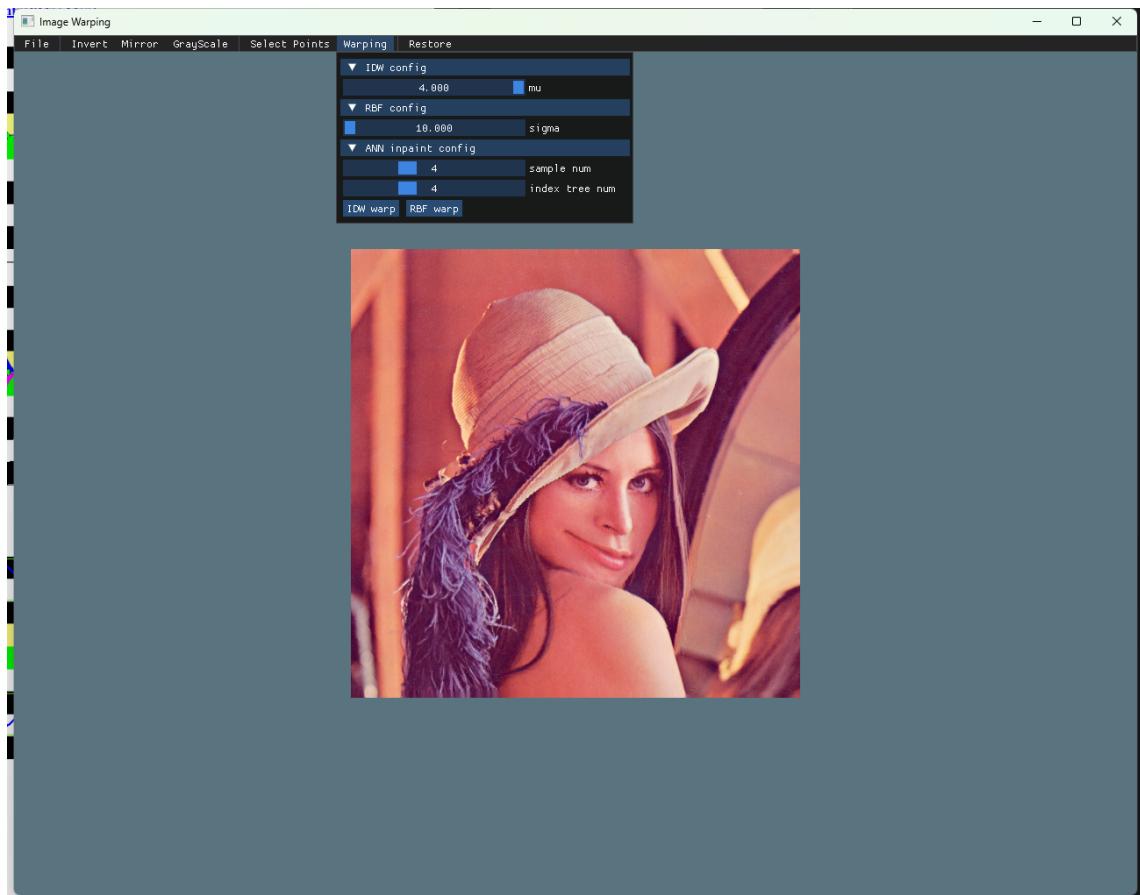
• $\mu = 4$



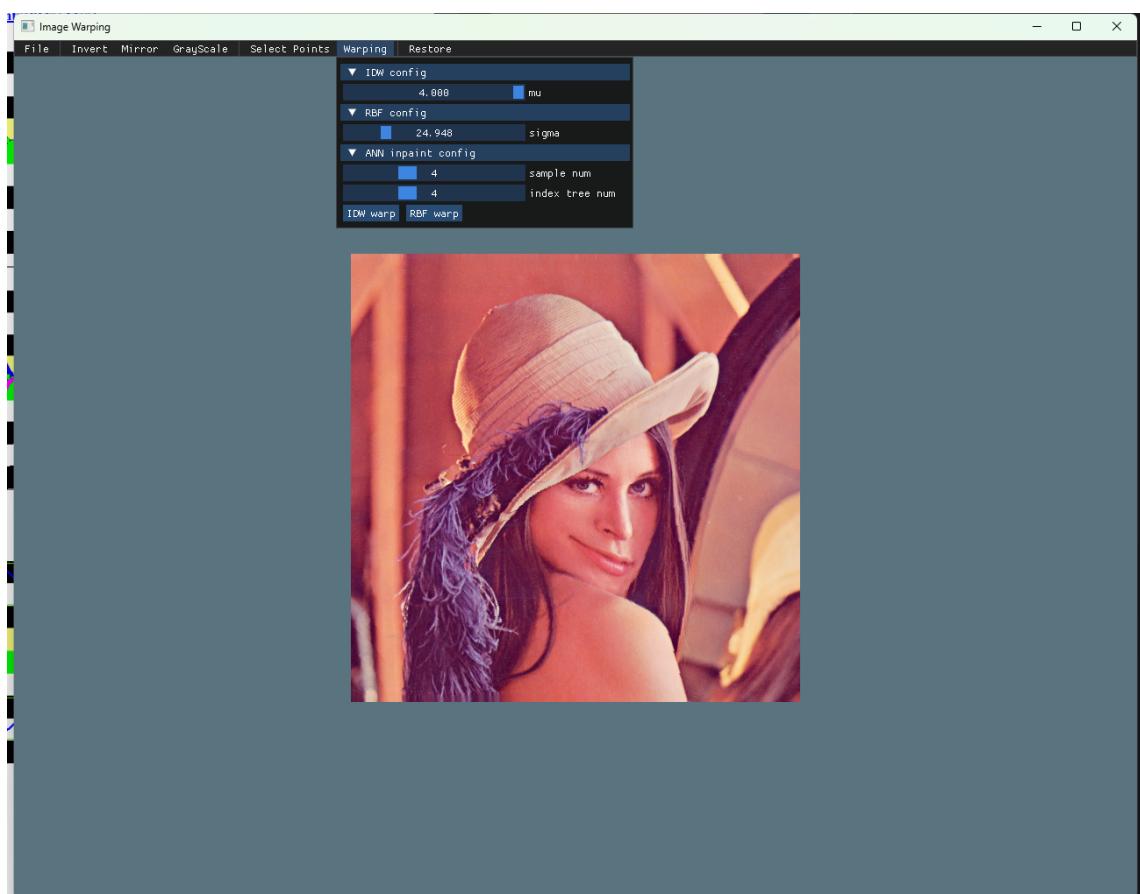
RBF方法

sigma即为径向基函数中的 r

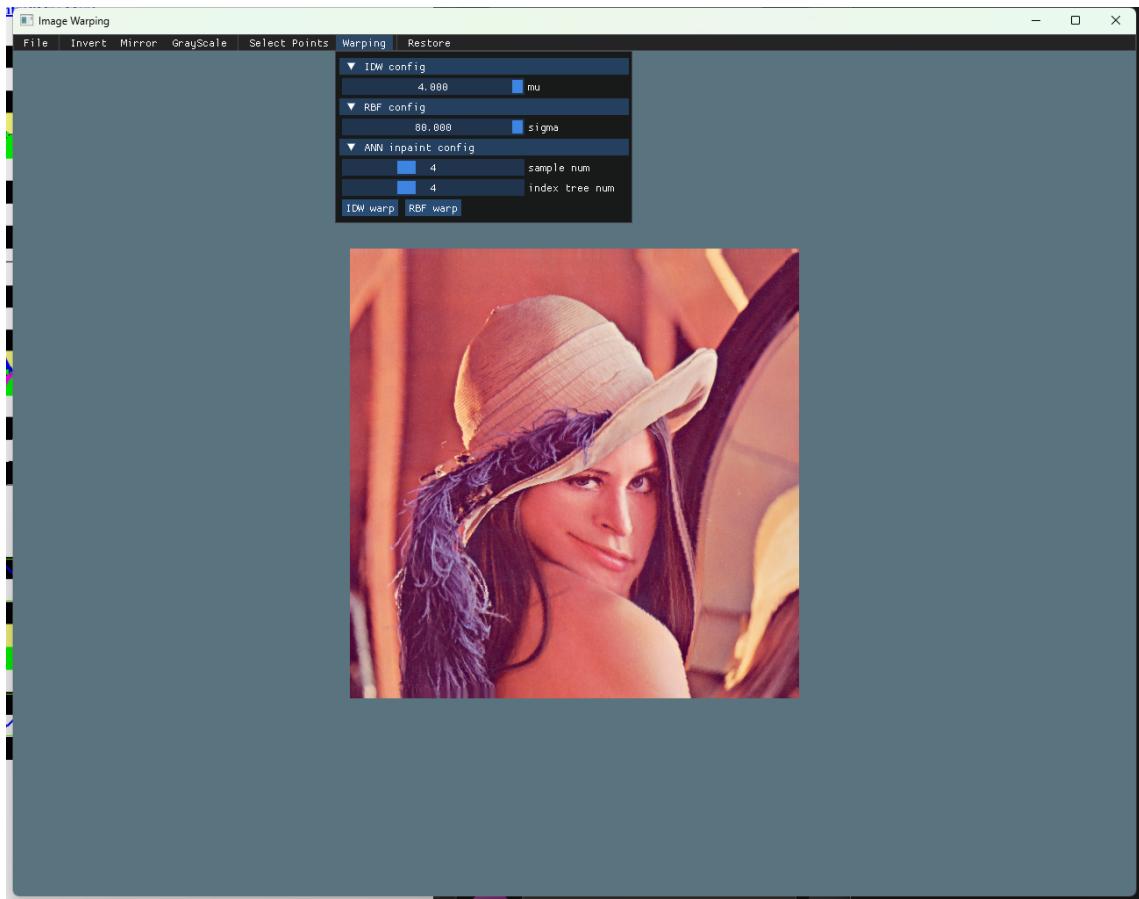
- $r = 10$



- $r = 25$



- $r = 80$



总结

通过上述演示不难发现：

- IDW方法的优点是简单易实现，缺点是对控制点的数量和位置敏感，容易产生过度变形的现象。
- RBF方法的优点是能够保持图像的平滑性和局部特征，缺点是计算量大，需要求解线性方程组。