山东大学 计算机科学与技术 学院

计算机视觉 课程实验报告

学号：201800130086	姓名： 徐鹏博	
实验题目：双边滤波		

实验过程中遇到和解决的问题：

双边滤波：计算权重时同时考虑空间位置和像素颜色之差

$$g(i,j)=\frac{\sum_{(k,l)\in S(i,j)} f(k,l)w(i,j,k,l)}{\sum_{(k,l)\in S(i,j)} w(i,j,k,l)}$$

$$w = ws * wr$$

$$ws = e^{(-\frac{(i-k)^2+(j-l)^2}{2\sigma_s^2})}$$

$$wr = e^{(-\frac{\|f(i,j)-f(k,l)\|^2}{2\sigma_r^2})}$$

获得高斯核:

```
double **Get_SpaceKernel(int size_d,double sigmaSpace) {
	double **SpaceKernel = new double *[size_d];
	for (int i = 0; i < size_d; ++i) {
		SpaceKernel[i] = new double[size_d];
	}
	double tmp = -2 * sigmaSpace * sigmaSpace;
	int kx, ky;
	kx = ky = size_d / 2;
	for (int i = 0; i < size_d; ++i) {
		for (int j = 0; j < size_d; ++j) {
			SpaceKernel[i][j] = exp(((i - kx) * (i - kx) + (j - ky) * (j - ky)) / tmp);
		}
	}
	return SpaceKernel;
}
```

获得颜色核:

```
double **Get_ColorKernel(double sigmaColor) {
	double **ColorKernel = new double *[256];
	for (int i = 0; i < 256; ++i) {
		ColorKernel[i] = new double[256];
	}
	double tmp = -2 * sigmaColor * sigmaColor;
	for (int i = 0; i < 256; i++) {
		for (int j = 0; j < 256; j++) {
			ColorKernel[i][j] = exp((i - j) * (i - j) / tmp);
		}

	}
	return ColorKernel;
}
```

双边滤波:

```
Mat Bilateral_Filter(const Mat &input,    double sigmaColor,double sigmaSpace) {
	Mat output;
	input.copyTo(output);
	int N = int(6 * sigmaSpace - 1) / 2; //一半长度
	int D = N * 2 + 1;//窗口总长度
	int H = output.rows, W = output.cols;
	int x, y;
	double **WeightColor = Get_ColorKernel(sigmaColor);
	double **WeightSpace = Get_SpaceKernel(D, sigmaSpace);
	double Value, NormalSum, Weight;
	for (int c = 0; c < 3; ++c) {
		for (int i = N; i < H - N; i++) {
			for (int j = N; j < W - N; j++) {
				Value = NormalSum = 0.0;
				for (int p = 0; p < D; ++p) {
					for (int q = 0; q < D; ++q) {
						x = i - p + N;
						y = j - q + N;
						Weight = WeightSpace[p][q] * WeightColor[input.at<Vec3b>(i,
j)[c]][input.at<Vec3b>(x, y)[c]];
						NormalSum += Weight;
						Value += input.at<Vec3b>(x, y)[c] * Weight;
					}
				}
				output.at<Vec3b>(i, j)[c] = Value / NormalSum;
			}
		}
	}
```

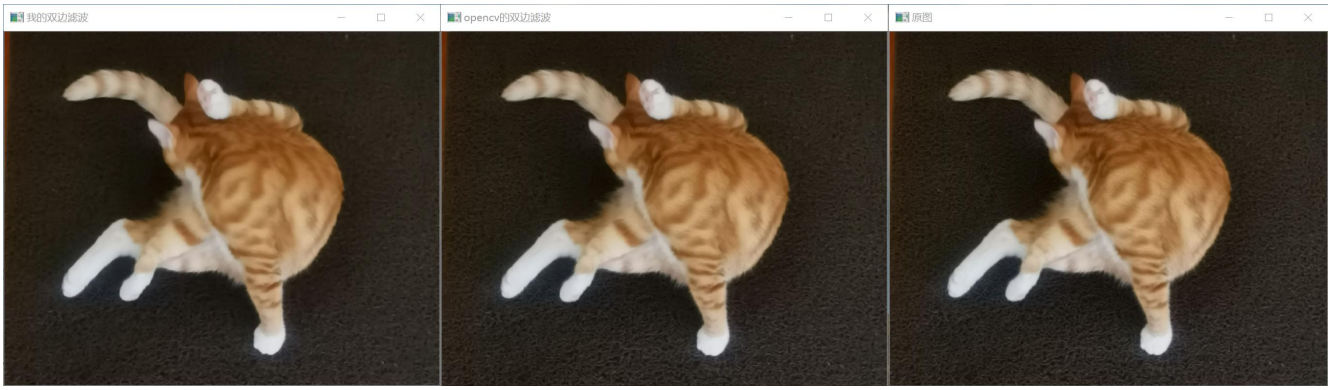
```
    return output;
};
```

和 opencv 内置双边滤波函数对比:

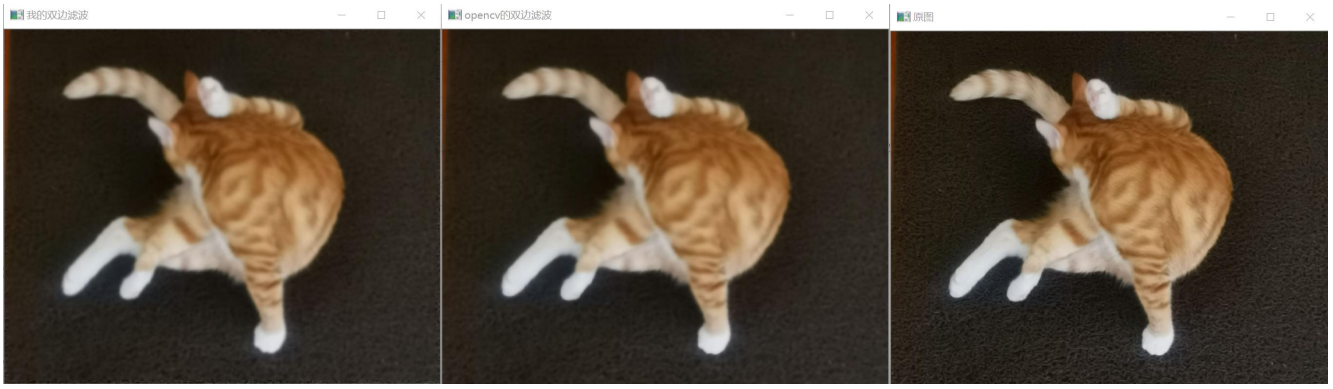
```
void Contrast(Mat input){
    Mat output1,output2;
    double start,end;

    string winName1="我的双边滤波",winName2="opencv 的双边滤波",winName0="原图";

    namedWindow(winName0, WINDOW_NORMAL);
    imshow(winName0,input);
    start=clock();
    output1 = Bilateral_Filter(input, 300, 5);
    end=clock();
    cout<<"my time:"<<double((end-start)/1000)<<"s"<<endl;


    namedWindow(winName1, WINDOW_NORMAL);
    imshow(winName1,output1);
    start=clock();
    bilateralFilter(input,output2,29,300,5,BORDER_DEFAULT);
    end=clock();
    cout<<"opencv's time:"<<double((end-start)/1000)<<"s"<<endl;
    namedWindow(winName2, WINDOW_NORMAL);
    imshow(winName2,output2);
}
```

效果图:

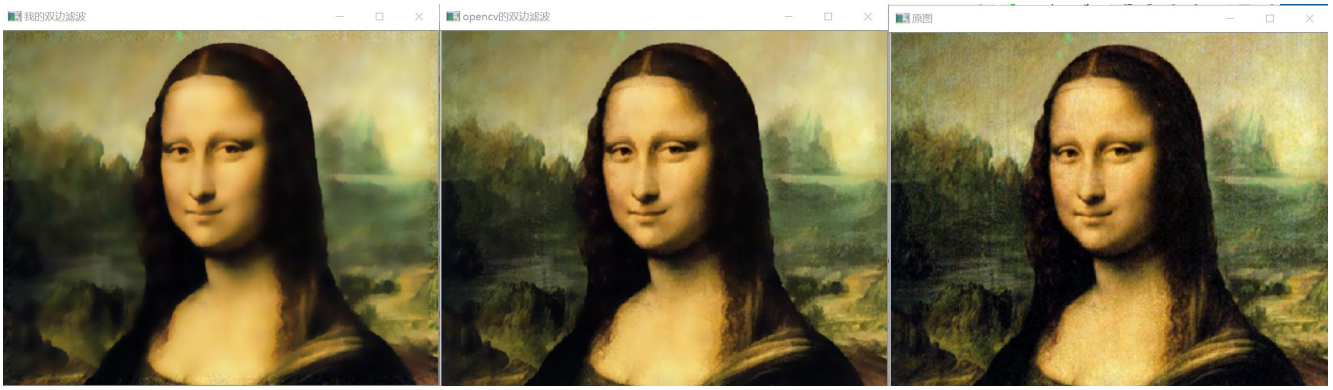
sigmaColor=10, sigmaSpace=5, Size=29



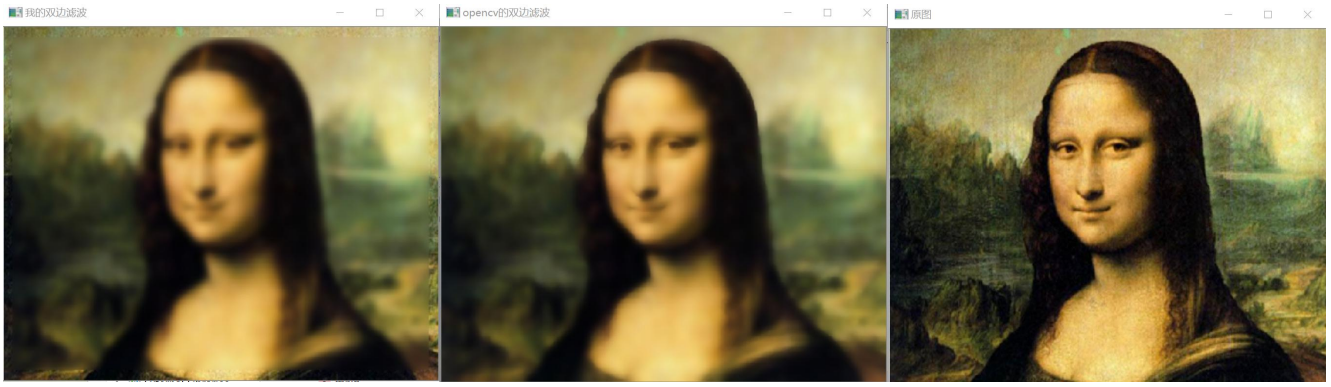
sigmaColor=300, sigmaSpace=5, Size=29



sigmaColor=30,sigmaSpace=5,Size=29



sigmaColor=300,sigmaSpace=5,Size=29



结果分析与体会：

1.当图像在非边缘区域时，邻域中的像素值差距相差不大，此时的双边滤波近似于普通的高斯滤波的效果，起到平滑效果。

2.当图像在边缘区域时，像素值相差很大。此时颜色与中心点差别小的领域点比重大，差别大的比重小，保持了边缘。

3.我的双边滤波运行时间和 opencv 的差别稍大，比如 Size=29 时蒙娜丽莎的微笑这个照片需要 14.066s 和 0.2s。