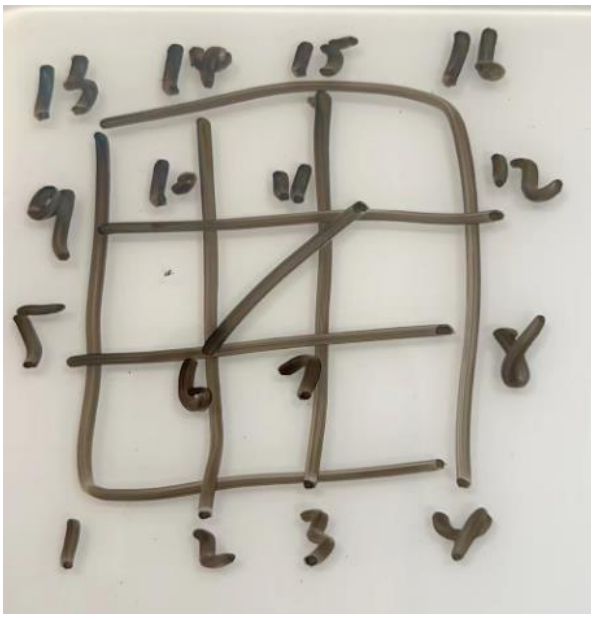
希望同学们通过本实验锻炼系统能力，实验具体模块如下：

1，功能模块：数据录入，数据结构、细分算法、可视呈现；
2，数据录入：obj 格式，实现基本的文件读写功能；
3，数据结构：就是一个图结构，点和边分别存储，需要用到查询一邻域的方法；
4，细分算法：Catmull 细分方法；
5，可视呈现：基于 opencv 绘制线段，在二维屏幕上呈现细分的迭代过程，或其他任何可视化手段（libigl, matlab，等等都可以）；

// 输入数据描述的是下面的图形



```
// 增加了面的索引
v 0.0 0.0
v 1.0 0.0
v 2.0 0.0
v 3.0 0.0
v 0.0 1.0
v 1.0 1.0
v 2.0 1.0
v 3.0 1.0
v 0.0 2.0
v 1.0 2.0
v 2.0 2.0
v 3.0 2.0
v 0.0 3.0
v 1.0 3.0
v 2.0 3.0
v 3.0 3.0
e 1 2
e 2 3
e 3 4
e 5 6
e 6 7
e 7 8
e 9 10
e 10 11
e 11 12
e 13 14
e 14 15
e 15 16
e 1 5
e 5 9
e 9 13
e 2 6
e 6 10
e 10 14
e 3 7
e 7 11
```

```
e 11 15
e 4 8
e 8 12
e 12 16
e 6 11

f 1 2 6 5
f 2 3 7 6
f 3 4 8 7
f 5 6 10 9
f 6 7 11
f 6 11 10
f 7 8 12 11
f 9 10 14 13
f 10 11 15 14
f 11 12 16 15
```

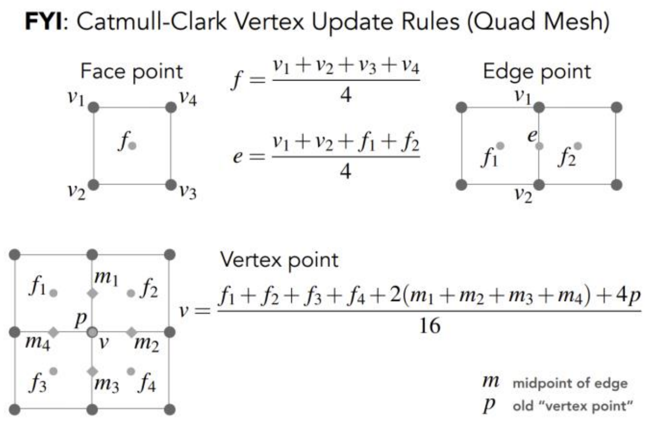
```
// 数据结构实例
Class V
{
Int v_id;
Double x;
Double y;
}
Class E
{
Int e_id;
Int V1;
Int V2;
}
Class F
{

Int f_id;
Vector<int> es; // 属于该面的所有边的索引
Vector<int> vs; // 属于该面的所有点的索引
}
```

在程序中存储数据：

```
Vector<V> vs; // 所有顶点
Vector<E> es; // 所有边
Vectors<F> fs; // 所有面
```

// 非奇异点的更新公式


FYI: Catmull-Clark Vertex Update Rules (Quad Mesh)

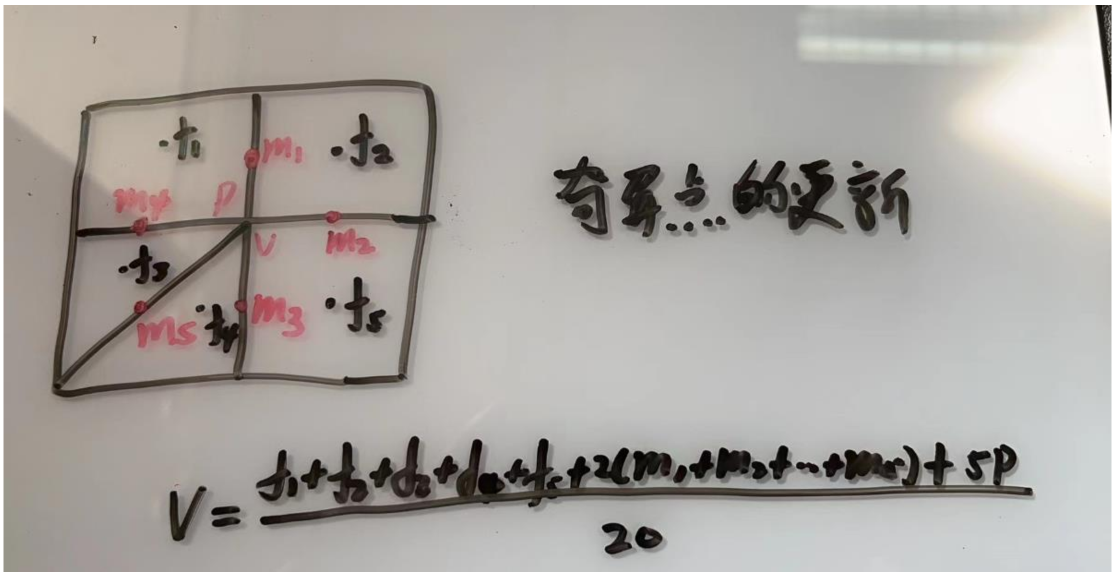
Face point $f = \frac{v_1+v_2+v_3+v_4}{4}$

Edge point $e = \frac{v_1+v_2+f_1+f_2}{4}$

Vertex point $v = \frac{f_1+f_2+f_3+f_4+2(m_1+m_2+m_3+m_4)+4p}{16}$

m midpoint of edge

p old "vertex point"


奇异点的更新

$$V=\frac{f_1+f_2+f_3+f_4+f_5+2(m_1+m_2+\cdots+m_5)+5p}{20}$$

// 奇异点的更新公式

// Catmal 细分算法的实现步骤 （ 每次迭代后网格的数据结构如何更新？）

1，生成所有的面点

2，生成所有的边点（边上的点）

3，更新已有顶点的位置

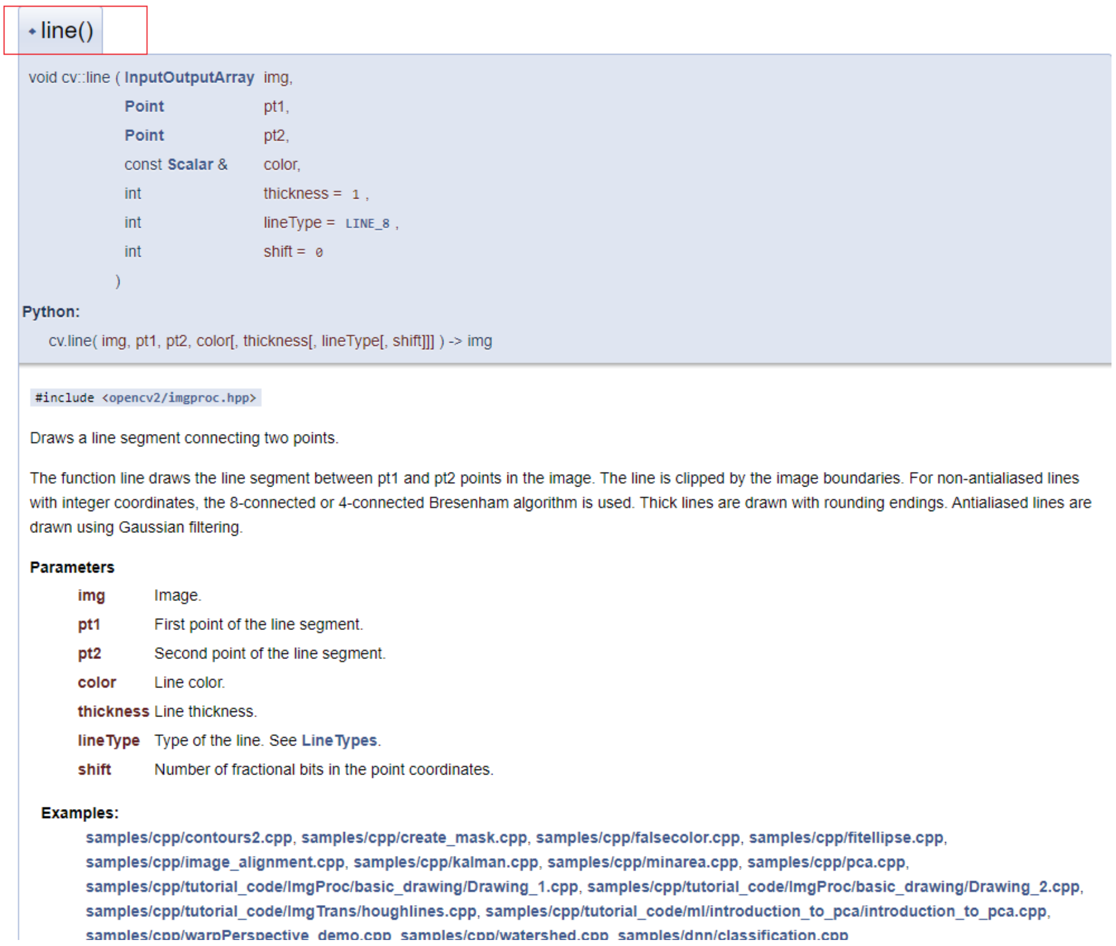
4，更新所有的边

4.1 删除旧的边：删除原来的所有边

4.2 添加新的边： 面点到边点的边， 连接边点到旧顶点的边

OpenCV 绘制**线段**和**点**

官方文档有相应的 line()以及 circle 函数



◆ line()

```
void cv::line ( InputOutputArray  img,
                Point             pt1,
                Point             pt2,
                const Scalar &    color,
                int               thickness = 1 ,
                int               lineType = LINE_8 ,
                int               shift = 0
              )
```

Python:

```
cv.line( img, pt1, pt2, color[, thickness[, lineType[, shift]]] ) -> img
```

`#include <opencv2/imgproc.hpp>`

Draws a line segment connecting two points.

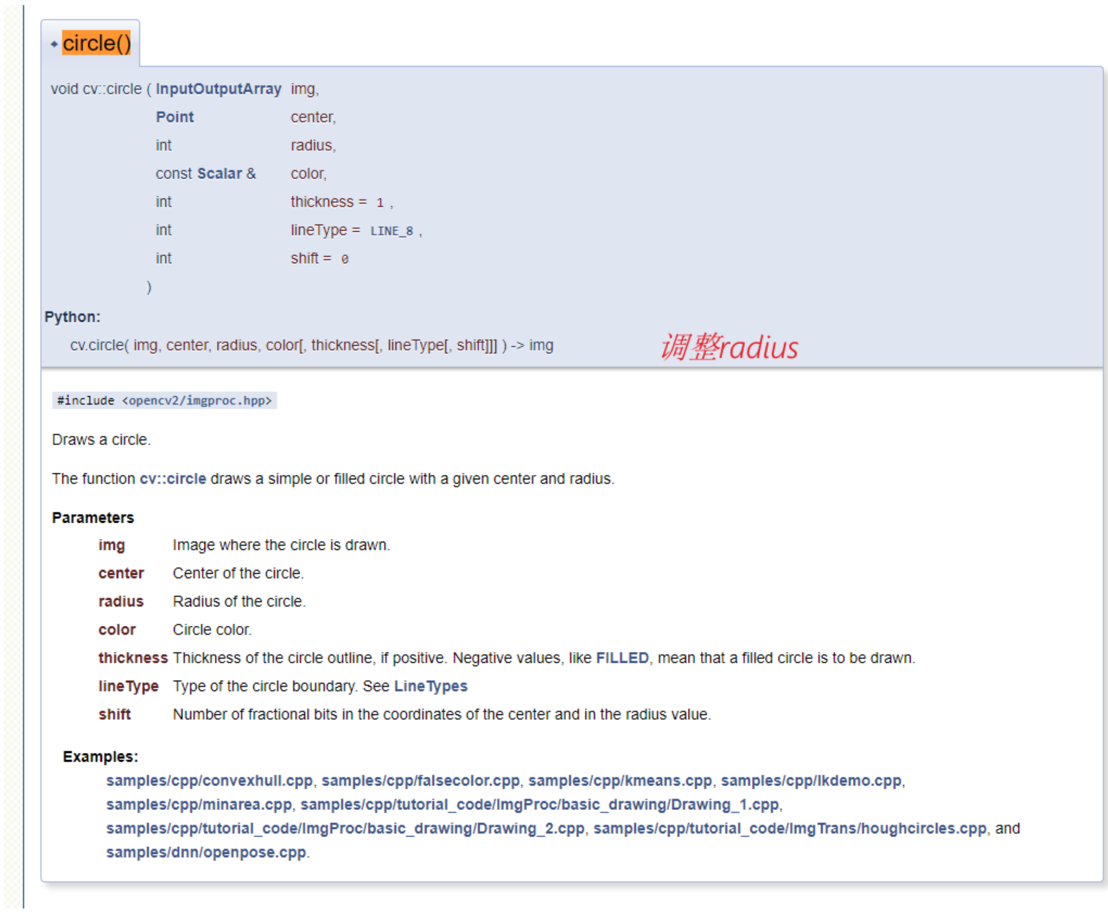
The function line draws the line segment between pt1 and pt2 points in the image. The line is clipped by the image boundaries. For non-antialiased lines with integer coordinates, the 8-connected or 4-connected Bresenham algorithm is used. Thick lines are drawn with rounding endings. Antialiased lines are drawn using Gaussian filtering.

Parameters

img	Image.
pt1	First point of the line segment.
pt2	Second point of the line segment.
color	Line color.
thickness	Line thickness.
lineType	Type of the line. See **LineTypes**.
shift	Number of fractional bits in the point coordinates.

Examples:

samples/cpp/contours2.cpp, samples/cpp/create_mask.cpp, samples/cpp/falsecolor.cpp, samples/cpp/fitellipse.cpp, samples/cpp/image_alignment.cpp, samples/cpp/kalman.cpp, samples/cpp/minarea.cpp, samples/cpp/pca.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_1.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_2.cpp, samples/cpp/tutorial_code/ImgTrans/houghlines.cpp, samples/cpp/tutorial_code/ml/introduction_to_pca/introduction_to_pca.cpp, samples/cpp/warpPerspective_demo.cpp, samples/cpp/watershed.cpp, samples/dnn/classification.cpp



◆ circle()

```
void cv::circle ( InputOutputArray  img,
                  Point             center,
                  int               radius,
                  const Scalar &    color,
                  int               thickness = 1 ,
                  int               lineType = LINE_8 ,
                  int               shift = 0
                )
```

Python:

```
cv.circle( img, center, radius, color[, thickness[, lineType[, shift]]] ) -> img
```

调整radius

`#include <opencv2/imgproc.hpp>`

Draws a circle.

The function **cv::circle** draws a simple or filled circle with a given center and radius.

Parameters

img	Image where the circle is drawn.
center	Center of the circle.
radius	Radius of the circle.
color	Circle color.
thickness	Thickness of the circle outline, if positive. Negative values, like **FILLED**, mean that a filled circle is to be drawn.
lineType	Type of the circle boundary. See **LineTypes**
shift	Number of fractional bits in the coordinates of the center and in the radius value.

Examples:

samples/cpp/convexhull.cpp, **samples/cpp/falsecolor.cpp**, **samples/cpp/kmeans.cpp**, **samples/cpp/lkdemo.cpp**, **samples/cpp/minarea.cpp**, **samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_1.cpp**, **samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_2.cpp**, **samples/cpp/tutorial_code/ImgTrans/houghcircles.cpp**, and **samples/dnn/openpose.cpp**.

demo：发在群里了 dra.zip