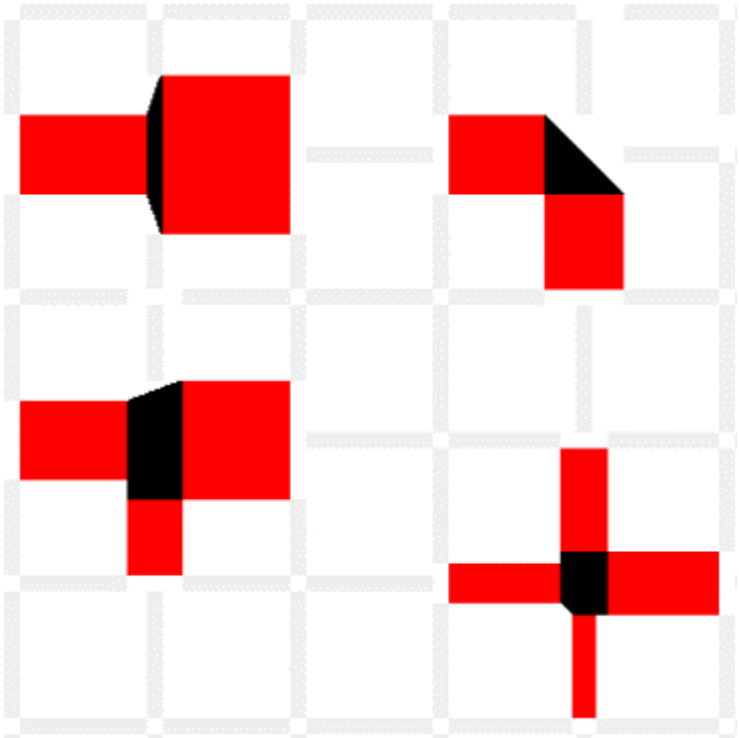
微流控生物芯片流体模拟计算界面

1 芯片绘制

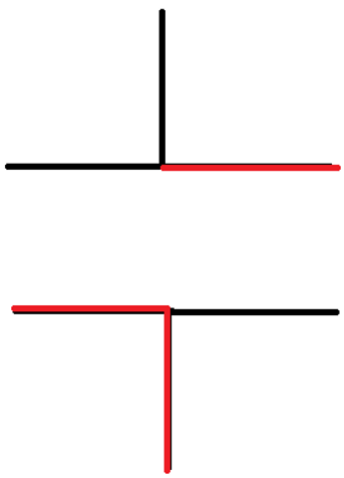
1.1 基础思想

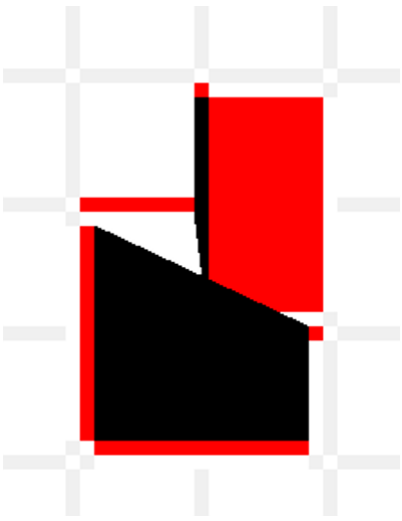
若相邻管道的宽度改变或出现拐角，则用斜边将它们连起来。



1.2 实现细节

如果仅仅这样实现，会出现问题。



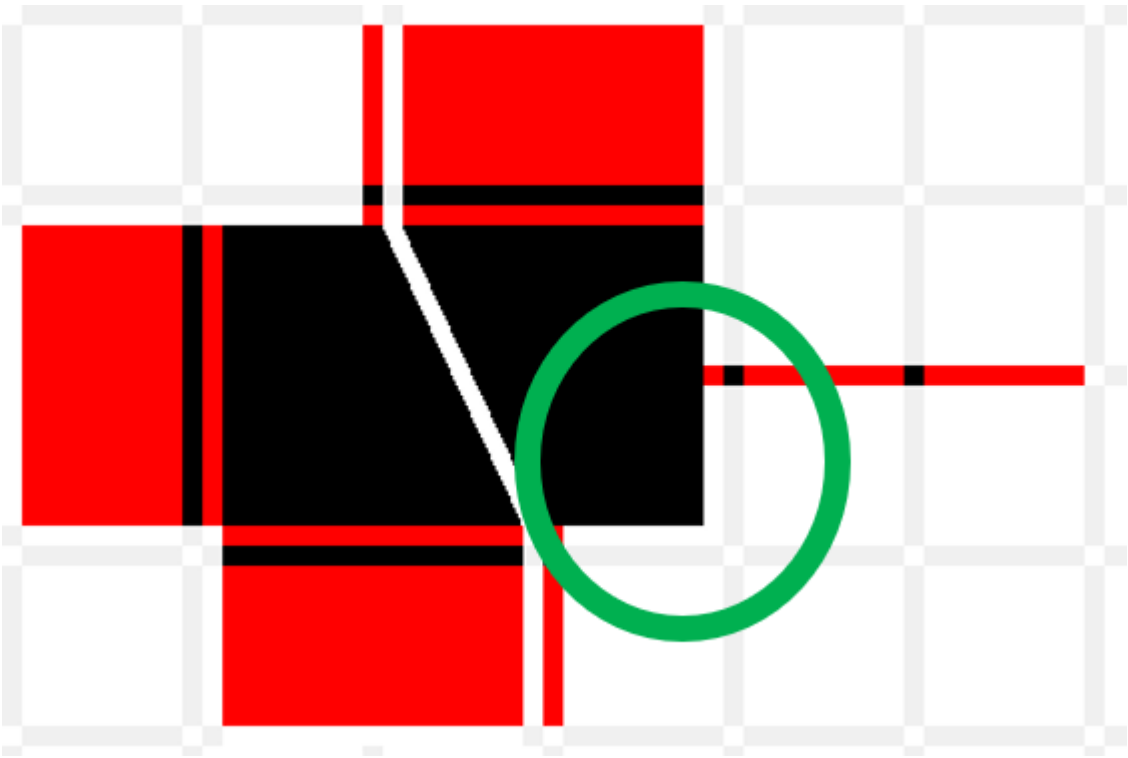


左图：
管子的绘制方式，
红色宽度为3000，
黑色宽度为200，
白色不存在

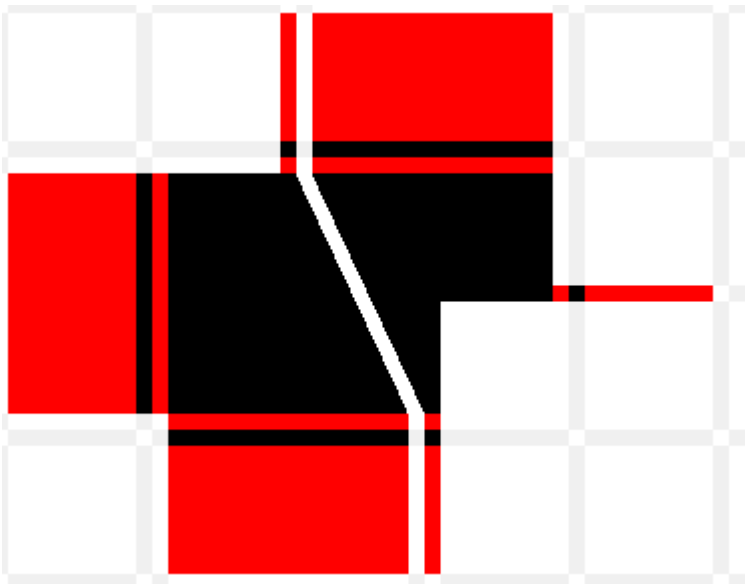
可以发现，右上方的管道与偏下的交叉点出现了重叠，其原因是连接左下方、下方的管子交叉点占用了太多空间，使得右上方管道无法正确绘制。

因此，在决定管道的绘制长度及交叉点的绘制方式时，不仅仅要考虑交叉点周围的四个管道的存在性及宽度，实际上要考虑以交叉点为中心$5*5$的范围内的管道的存在性与宽度。若在$5*5$的范围内存在较粗的管子，则需要缩短部分管子并将交叉点延伸，比如上图需要将上方的交叉点向右延伸，并缩短右上方的管子的长度。

但是这样又引入了新的问题。



上图绿色圆圈中的部分，本不应被绘制，但由于我将交叉点向下进行了延伸，这部分也被绘制为了管子。为解决这个问题，绘制交叉点时，需要同时考虑向上下左右的延伸及上下左右实际连接的管子的宽度。下图是我最终绘制的效果。



以上是我的绘制方法的主要原理，下面是一些我的绘制算法的部分运行结果。

按照我的绘制算法，宽度的变化较为自然，并且能够保证原本不相连的管道在进行宽度调整后依然不相连。

1.3 代码实现

绘制算法用`ChipShower`，`PipeWidget`,`InterWidget`三个类进行实现。

`ChipShower`类负责综合管道信息，告知`PipeWidget`所需绘制的管道长度，告知`InterWidget`类绘制连接点所需要的信息。

`PipeWidget`类负责管道的绘制，`InterWidget`类负责连接点的绘制。

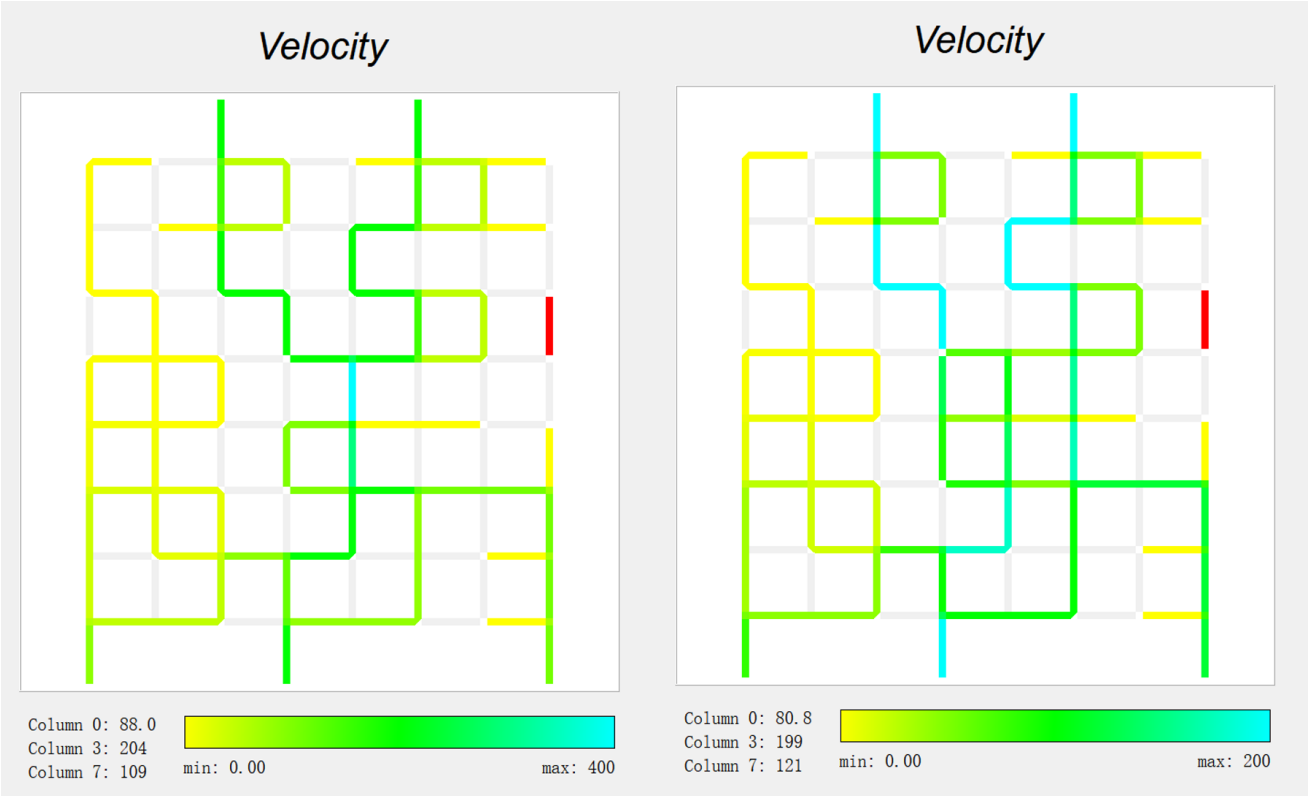
1.4 颜色显示

管道共有四种状态。

状态描述	**`PipeShower::Status`中对应的值**	**绘制颜色**
正常存在	`exist`	渐变色
存在但不与输入输出管道相连	`notLink`	红色
存在但流速极小可忽略不计	`unKnown`	淡粉色
不存在	`notExist`	浅灰色

渐变色是指从黄色到绿色到蓝色的线性渐变，其渐变过程如下：

其中，黄色代表较小值，蓝色代表较大值，计算界面中显示了最小值，最大值所对应的具体数值。根据管道的数值的取值范围不同，具体配色方案也会相应改变。下图是两个构成接近的管道的绘制效果。



2 模拟计算

实现了对输出管道的流速及浓度的计算，对于一个大小为$n*n$的芯片，流速计算的复杂度为$O(n^6)$，浓度计算的复杂度为$O(n^2)$

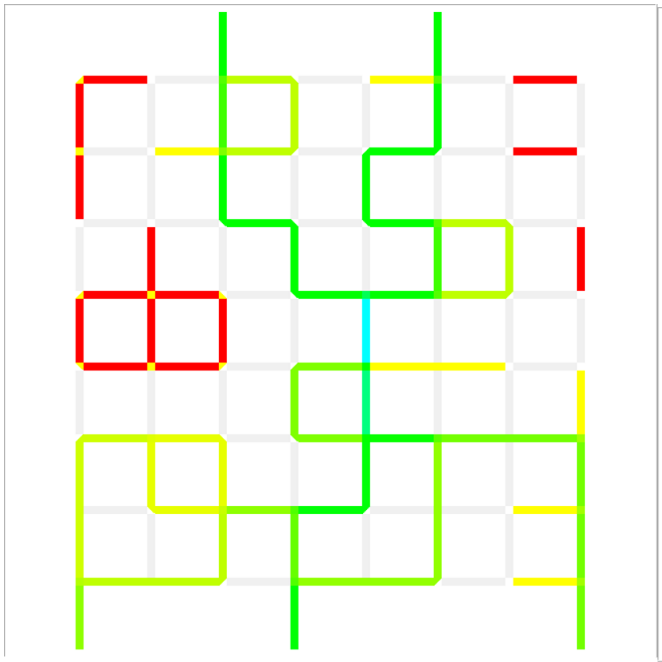
2.1 流速模拟

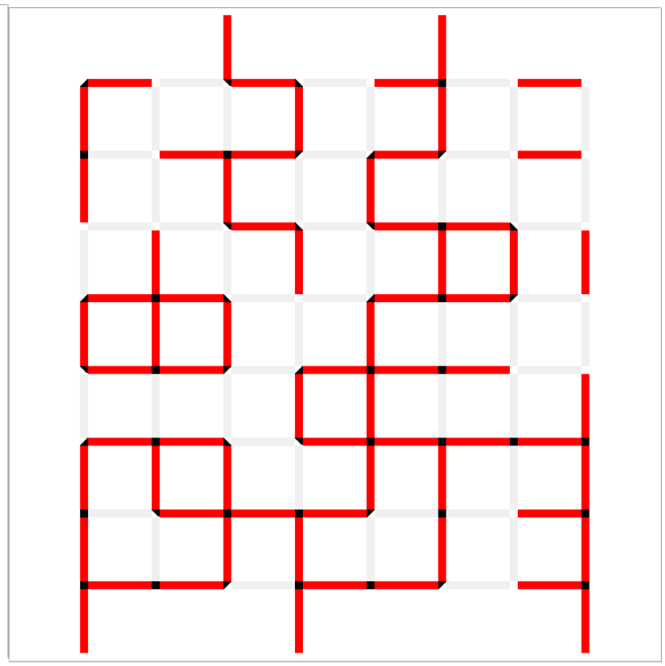
将管道视为电阻，输入管道视为恒流电源，流速视为电流，则可以使用基尔霍夫电路定律对流速进行计算。首先计算每个交叉点的电势ϕ，对交叉点x，设其周围4个节点的电势分别为ϕ_i（$i\in[1,4]$），四个管道的长度分别为$l_i(i\in[1,4])$，则根据交叉点的输入流量等于输出流量可知电势满足下述方程：

- 若X为输出节点，$\phi_x=0$
- 若X为输入节点，$\sum_{i=1}^{4}(\phi_i-\phi_x)/l_i=1$
- 否则，$\sum_{i=1}^{4}(\phi_i-\phi_x)/l_i=0$

  通过这n^2个线性无关的方程，可以解出每个点的电势，再通过每个管道的电势差与长度计算管道的流速。

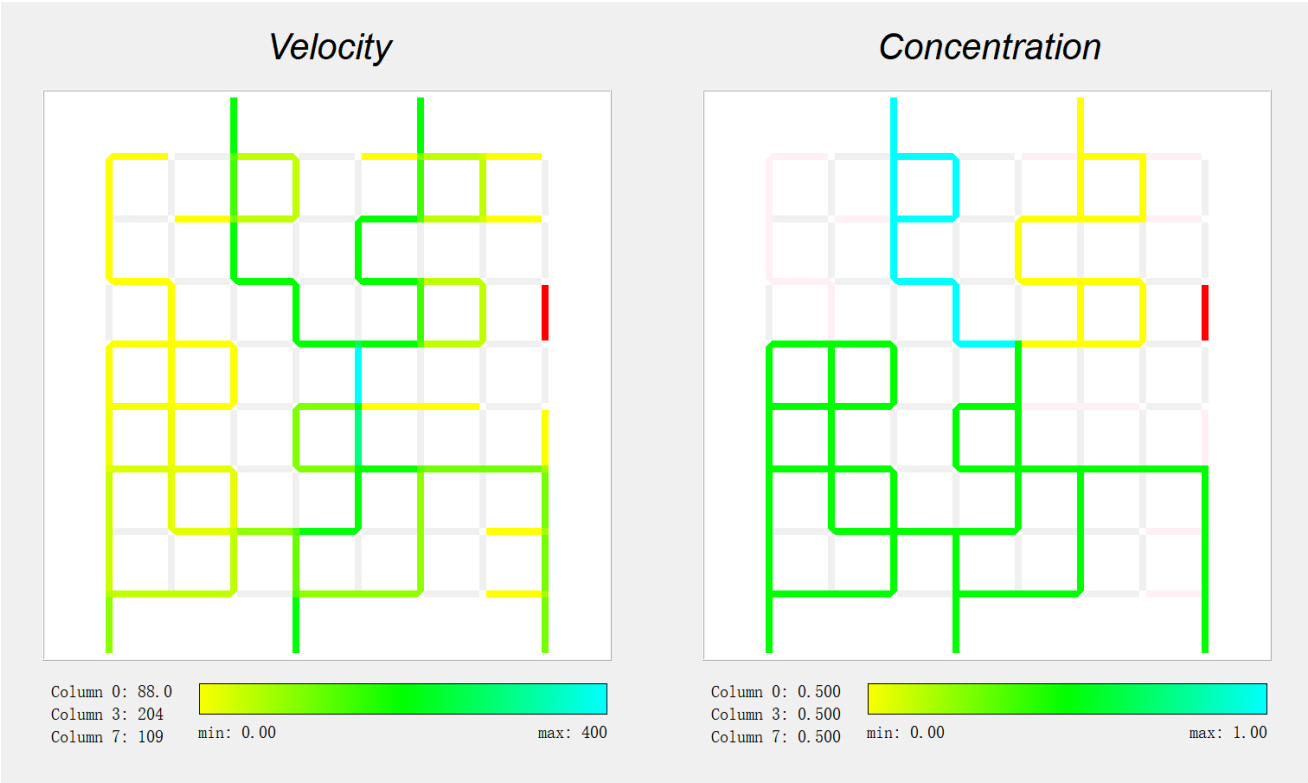
  若部分管道不与输入输出管道相连，则需要在计算之前将它们剔除，若输入输出管道不连通，则不进行计算





2.2 浓度模拟

浓度模拟使用拓扑排序，按照由入到出的顺序依次计算每个管道的浓度。由于流速模拟会产生一定的精度误差，浓度模拟时需要首先提出流速极小的管道。



2.3 程序接口

流速模拟与浓度模拟在类`Calculator`中实现(`RawChip`为存储芯片结构的类，3.5节将对其结构进行介绍)

```
class Calculator
{
public:
    Calculator(const RawChip* cur);
    void setCurChip(const RawChip* cur);
    const std::vector<double>& getV() const;
    const std::vector<double>& getC() const;
};
```

通过构造函数`Calculator(const RawChip* cur)`或成员函数`setCurChip(const RawChip* cur)`向类中传递一个芯片的结构，该类可计算出这个芯片的各个管道的流速与浓度，计算结果可通过分别通过函数`const std::vector<double>& getV() const;`与`const std::vector<double>& getC() const;`得到，该类的使用示例如下：

```
Calculator calc(&newChip);
velo = calc.getV(); //获取流速
con = calc.getC(); //获取流量
```

3 芯片修改

实现了管道的插入、删除、宽度修改、信息展示操作。

3.1 管道插入

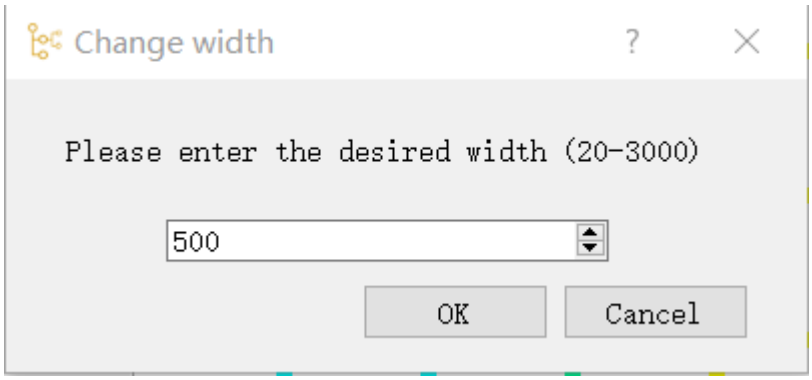
左键点击空白位置，可在该位置加入一个宽度为默认值的管道，若无法加入则不进行响应。右键点击空白位置，可直接设定插入管道的宽度。

3.2 管道删除

左键点击存在的管道，可以将其删除.

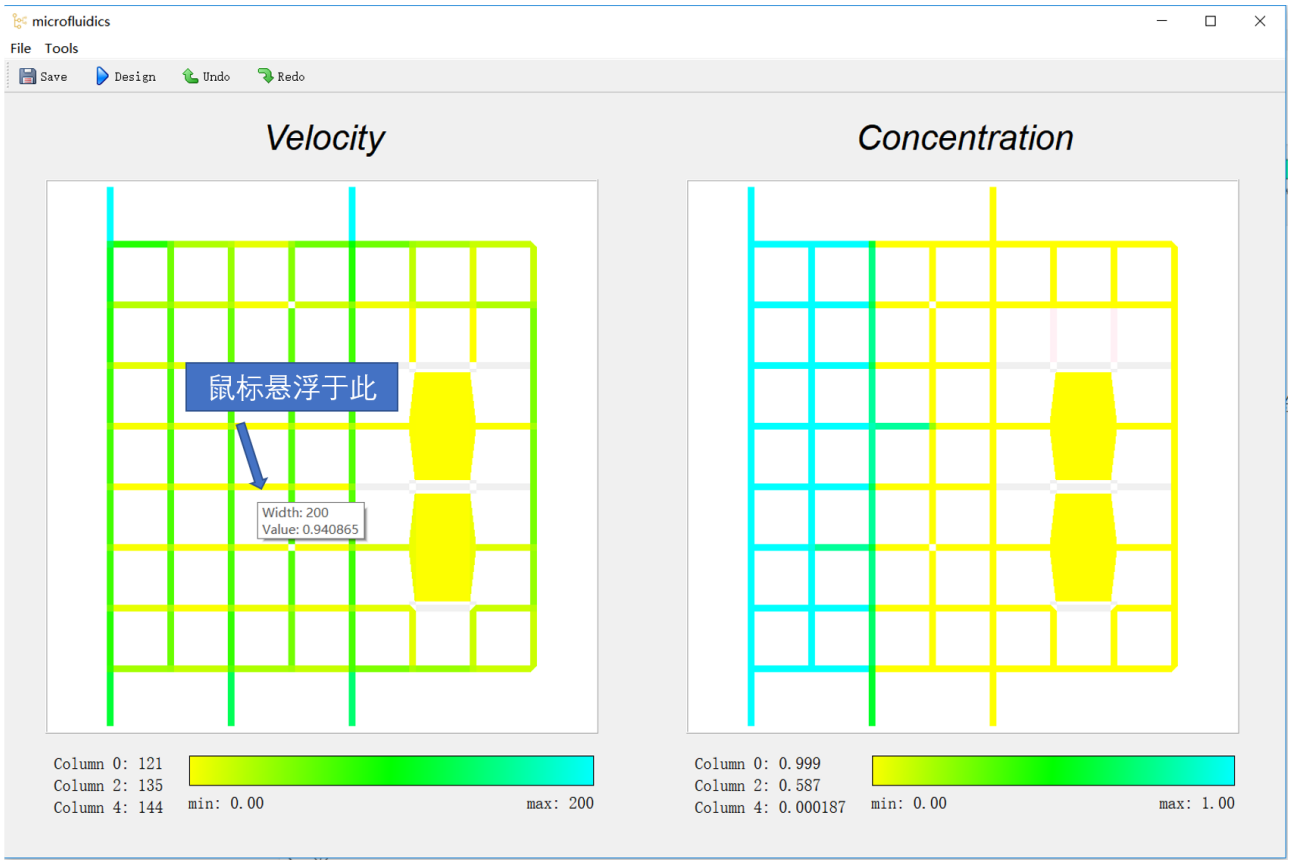
3.3 宽度修改

右键点击某个管道的位置（存在与不存在皆可），可在弹出对话框中设置该管道的宽度，宽度必须符合"对于同一列的管道，任意相邻管道之间的距离不能小于初始时管道的宽度"这一要求，否则无法输入。



3.4 管道信息显示

若将鼠标悬浮于管道之上，可得到该管道的宽度及流速或浓度的信息。



3.5 芯片定义

芯片通过`struct RawChip`与`class Chip`两个类定义。

`RawChip`存储了芯片的基本构造，包括芯片大小`n`，输入管道所在列`input`，输出管道所在列`output`，各个管道的宽度`width`，允许进行拷贝，并提供了一些查询管道、节点之间关系的函数。同时，还重载了`<<`运算符用以支持芯片的导入与导出

```
struct RawChip
{
    int n;
    QVector<double> width;
    QVector<int> input;
    QVector<int> output;
    bool validID(int id) const;
    int getType(int id) const; //0 竖着的 1 横着的 2 input 3 output
    int X(int id) const;
    int Y(int id) const;
    int getID(bool isVer, int x, int y) const;
    bool isVertical(int id) const;
    double UpLeftWidth(int id) const;
    double UpRightWidth(int id) const;
    double DownLeftWidth(int id) const;
    double DownRightWidth(int id) const;
    double upWidth(int x, int y) const;
    double downWidth(int x, int y) const;
    double leftWidth(int x, int y) const;
```

```
    double rightWidth(int x, int y) const;
    int fromVector(const QVector<int> vec, int id) const;
    int leftID(int x, int y) const;
    int rightID(int x, int y) const;
    int upID(int x, int y) const;
    int downID(int x, int y) const;
    int getWidth(int id) const;
    int getMaxWidth(int id) const;
    bool canChange(int id) const;
    QVector<int> getConnectivity() const;
    bool ioLink() const;
};
QDataStream& operator << (QDataStream &os, const RawChip &chip);
QDataStream& operator >> (QDataStream &is, RawChip &chip);
```

`Chip`类是`QObject`的子类，提供了一些与芯片进行交互的函数，如芯片生成`generate`，芯片载入`load`，芯片导出`print`，加入管道`addPipe`，删除管道`deletePipe`，改变宽度`changePipe`，同时还可通过`value`与`valueDir`获取芯片状态，通过`setValue`改变芯片。

```
class Chip : public QObject
{
    Q_OBJECT
public:
    explicit Chip(QObject *parent = nullptr);
    enum generateType
    {
        empty=0, random=90, full=100
    };
    static const int inputPipe = 2;
    static const int outputPipe = 3;
    static const int pipeLength = 1800;
    static const int pipeWidthDefault = 200;
    static const int pipeMinimalWidth = 20;
    static RawChip generate(generateType type, int n, QVector<int> inputPipes,
QVector<int> outputPipes);
    static RawChip load(QString path);
    void print(QString path);
    RawChip addPipe(int id);
    RawChip deletePipe(int id);
    RawChip changePipe(int id, int width);
    RawChip value() const { return c; }
    const RawChip* valueDir() const { return &c;}
    void setValue(const RawChip& cc) { c = cc; }

private:
    RawChip c;
};
```

3.6 管道定义

管道定义为`PipeWidget`类。该类定义了一系列前缀为“set”的成员函数，用户可通过这些函数对管道进行设置。该类重写了鼠标点击事件，用以实现对单击/双击的响应，并重写了`event`函数以响应鼠标悬浮。

```
class PipeWidget : public QWidget
{
    Q_OBJECT
public:
    enum Status { exist, notExist, notLink,unKnown };
    explicit PipeWidget(QWidget *parent = nullptr);
    void setWidthSaver(int* saver);
    void setColor(double rate);
    void setMaxWidth(int w);
    void setVertical(bool v) { isVertical = v; }
    void setStatus(Status s) { status = s; }
    void setCanChange(bool c) { canChange = c; }
    void setRealData(int width, double value);

protected:
    void paintEvent(QPaintEvent *e);
    void mousePressEvent(QMouseEvent *e);
    bool event(QEvent *event);

signals:
    void inserted();
    void deleted();
    void changed();

public slots:
    void insertDelete();
    void changeWidth();
};

```

若对管道进行插入、删除、改变宽度操作，管道会分别发出`inserted`，`deleted`，`changed`信号，用以与父组件进行交互。该类的父组件为`ChipShower`类，主要功能是对每个管道的显示位置与显示方式进行规划，后面将详细介绍该类的定义。`ChipShower`收到管道状态变化的信号后，会将发生改变的管道的编号及变化方式通过信号槽发送至`MainWindow`，`MainWindow`接收到信号后，更新其所存储的芯片的管道信息，调用`Calculator`的相关函数对新管道的流速、浓度信息进行重新计算，然后调用Chipshower中的函数更新芯片显示界面。

4 基本操作

为方便用户使用，实现了当前芯片的导入与导出，管道操作的撤销与重做，自动生成一定形式的芯片及设计给定浓度的芯片的功能。

4.1 导入与导出

可将芯片存储为以".chip"为后缀的二进制文件，也可读入一个".chip"文件，绘制其存储的芯片。".chip"文件中存储了芯片的节点数，输入输出管道所在列，管道宽度，读入后需要重新调用`Calculator`与`ChipShower`对芯片进行绘制。导入与导出分别在`MainWindow`的如下两个函数中实现。

```
    void on_exportChipAction_triggered();
    void on_loadChipAction_triggered();
```

4.2 撤销与重做

利用`QUndoStack`对管道修改的历史信息进行存储。每个操作存储于一个`MyUndoCommand`的实例中。`MyUndoCommand`是`QUndoCommand`的派生类，重写了`undo`，`redo`两个函数，存储了每一次操作前后芯片的状态，其监控的芯片的地址。`MyUndoCommand`是一个模板类，T是监控的类的名称，Value是监控的类中实际存储信息的类的名称。在本程序中，`T`为`Chip`，`Value`为`RawChip`

```
template<class T, class Value>
class MyUndoCommand : public QUndoCommand
{
public:
    MyUndoCommand(T& target, Value& newItem, QUndoCommand *parent = 0):
        QUndoCommand(parent), m_target(target), m_pre(target.value()),
m_cur(newItem) {}
    void undo()
    {
        m_target.setValue(m_pre);
    }
    void redo()
    {
        m_target.setValue(m_cur);
    }

private:
    T& m_target;
    Value m_pre, m_cur;
};
```

4.3 自动生成

可自动生成空芯片、完整芯片，及按每个管道存在概率为90%生成新芯片。`MainWindow`获取用户所需的芯片种类后，调用`Chip::generate`生成芯片。该函数通过`Chip::generateType`获取用户所需芯片类型，`n`，`inputPipes`，`outputPipes`获取用户所需芯片形态，返回一个按用户要求生成的`RawChip`，下面为Chip类相关的定义。

```
enum generateType
{
    empty=0, random=90, full=100
};
```

```
static RawChip generate(generateType type, int n, QVector<int> inputPipes,
QVector<int> outputPipes);
```

4.4 自动设计

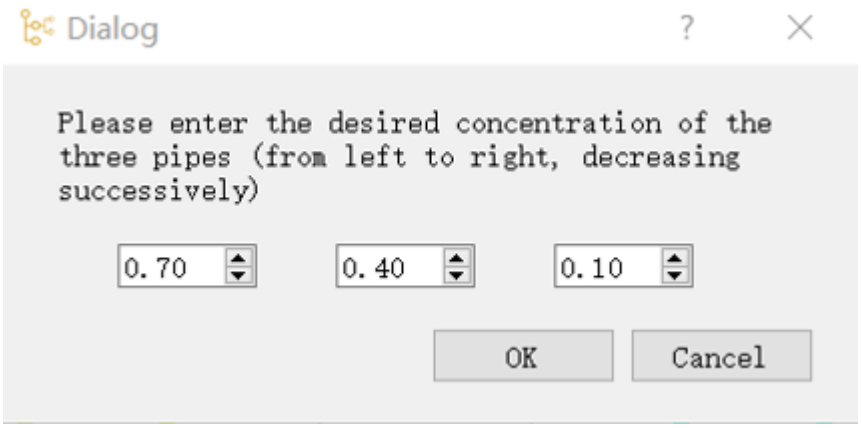
可按照用户所需要的输出浓度自动设计芯片，自动设计过程定义在类`ChipDesigner`中。该类可通过构造函数传入芯片的输出形态，调用`design`进行设计，调用`loss`计算设计的芯片与目标值的误差。

```cpp
class ChipDesigner
{
public:
    ChipDesigner(const RawChip& c, int randRound_ = 10, int simuRound_ = 1000,
                 double T_initial_ = 0.0001, double dropSpeed = 0.99, unsigned int
seed_ = 0);
    RawChip design(const QVector<double>& tar);
    double loss(const std::vector<double>& result);
};
```

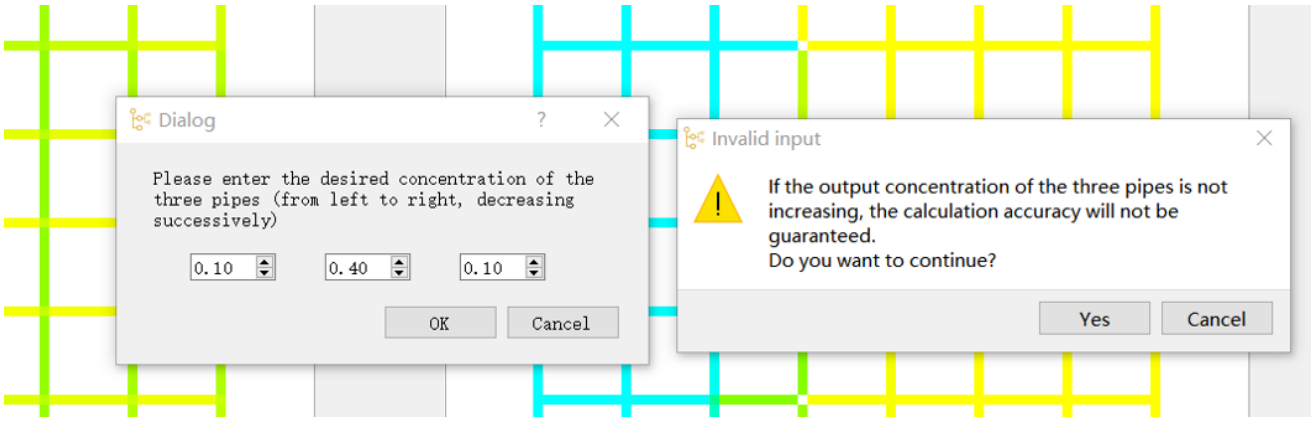
loss定义为三个管道的误差的平方和，这样可以使计算更关心误差较大的管子。

芯片设计的算法为随机生成10个芯片，选取其中loss最小的一个，再运行1000轮的模拟退火。

用户点击“自动设计”的按钮后，会弹出一个对话框，用以输入所需浓度。



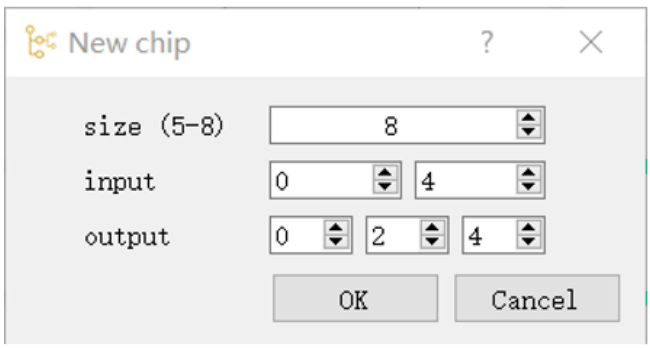
由于左边的输入管道浓度大，右边的输入管道浓度小，最终三个管道的输出浓度会满足从左至右单调不降的关系，若用户输入不满足此关系，则进行警告。

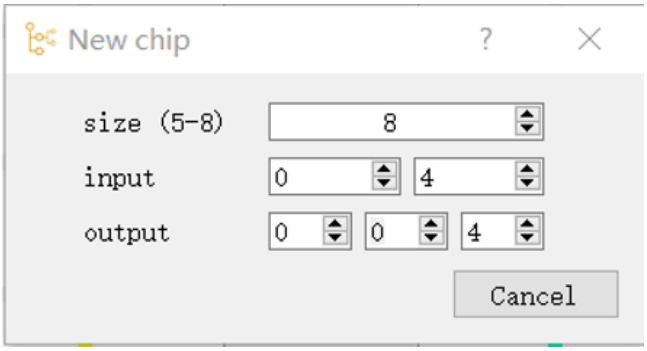


5 细节处理

5.1 新建芯片的合法性判定

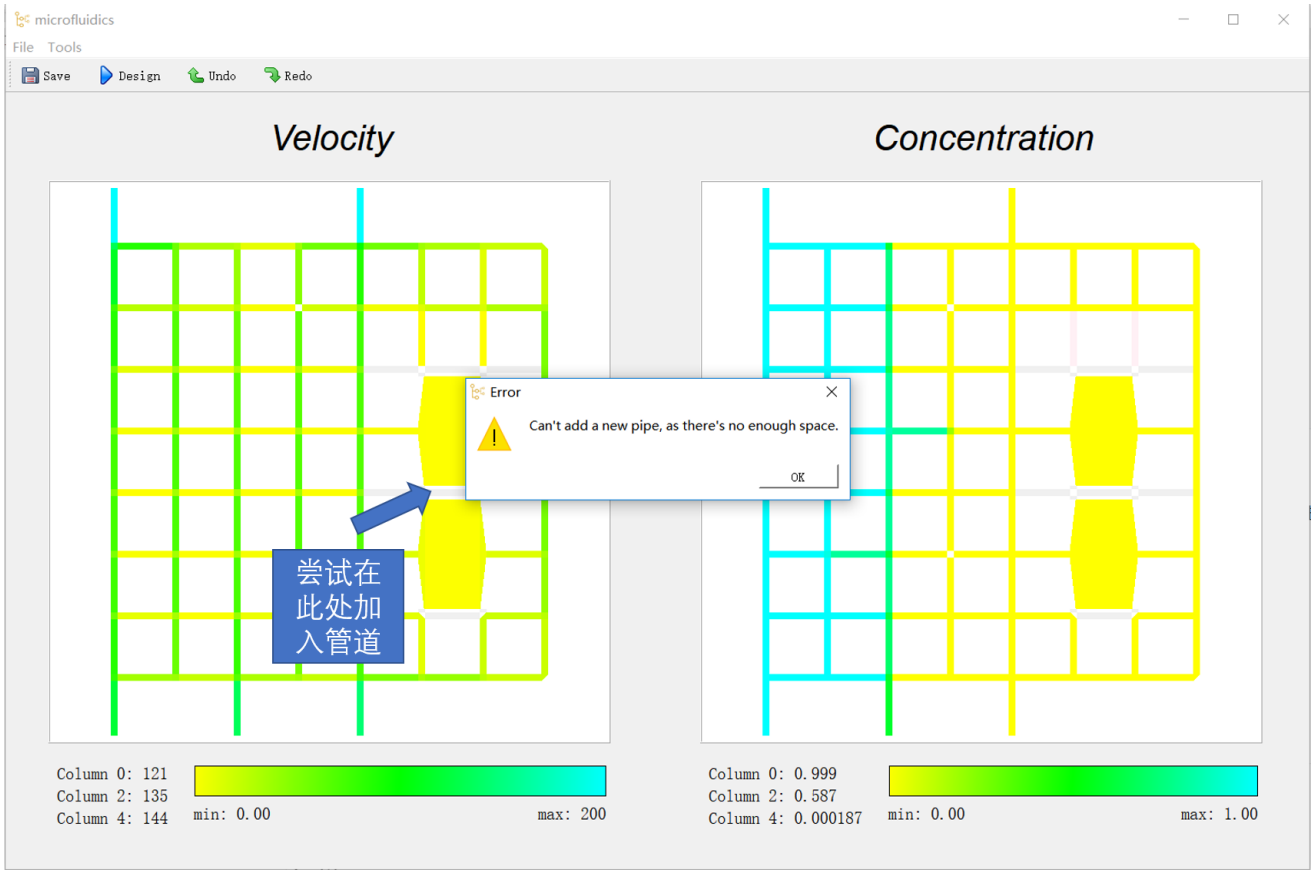
若输入非法，则将确定键隐藏。下方左图为合法输入，右图为非法输入。





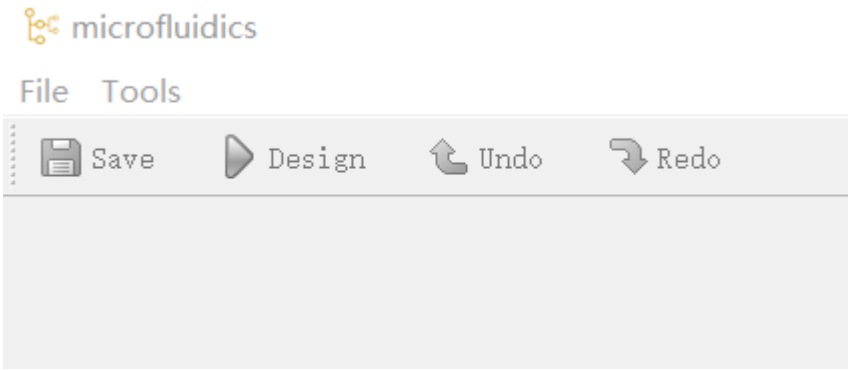
5.2 插入管道的可行性判定

若在一个由于空间过小无法添加管道的地方点击左键添加，则不进行响应，若点击右键添加，则弹出对话框提示不可添加。



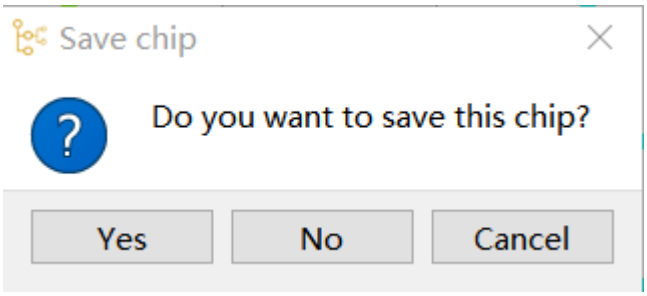
5.3 无效按键变灰

初始时，界面中没有芯片，因此无法进行保存、撤销、重做等操作，在这种情况下将按钮设为灰色。



5.4 保存提醒

在关闭界面、新建芯片、导入芯片的时候询问是否对现有芯片进行保存。



5.5 快捷键

按键	功能
Alt+Shift+N	新建空芯片
Ctrl+Shift+N	新建随机芯片
Ctrl+N	新建完整芯片
Ctrl+Alt+N	导入已有芯片
Ctrl+S	保存
Ctrl+D	自动设计
Ctrl+Z	撤销
Ctrl+Shift+Z	重做

5.6 图标

设置了软件图标和一些功能的图标，下面是一些用到的图片