浙江大学

课程名称：计算机动画

姓　　名：猜猜

学　　院：计算机学院

专　　业：数字媒体技术

学　　号：

指导教师：

2019 年 11 月 2 日

浙江大学实验报告

课程名称：计算机动画 实验类型：综合

实验项目名称：关键帧动画系统

学生姓名：猜猜 专业：数字媒体技术 学号：

同组学生姓名：无 指导老师：

实验地点： 实验日期：2019 年 11 月 2 日

一、 实验目的和要求

1.掌握线性插值和矢量线性插值的思路和算法，了解两种变形算法不同的性能和对变形过程中对于图形形状的影响。

2.掌握并实践线性插值和矢量线性插值的具体思路和程序代码的对应关系。

3.了解关键帧动画系统的结构，了解动画动态控制的基本原理和方法，提高相关动画编程能力。

二、 实验内容

关键帧动画技术是计算机动画中的一类重要技术。本实验选取线性插值和矢量线性插值作为实验内容，体现关键帧动画系统的结构，变形算法的思想以及不同算法对应的不同性能。设计利用两种变形算法体现变形的过程，即绘制出开始的图形及结束的图形，体现变化过程中的图形。

三、 实验器材

Windows 10

QT 5.12.0

四、 实验原理与过程分析

1. 线性插值和矢量线性插值的具体思路与程序代码的对应关系

（1）线性矢量插值的具体思路

线性插值，也就是给定初始点集合和终止点集合，然后给定一个映射关系。对一一对应的点的位置，即对 x,y 坐标进行线性插值。过程中的坐标值如下所示：

$$x = x0*t + x1*(1-t);$$

$$y = y0*t + y1*(1-t);$$

相关代码如下：

```
1. float v = (float)this->time / this->number;
2. for(int i=0; i < this->begin_shape.point.size(); i++){
3.     this->process_shape.addPoint((1-v)*this->begin_shape.point[i] + v*this->
   end_shape.point[i]);
4. }
```

（2）矢量线性插值的具体思路

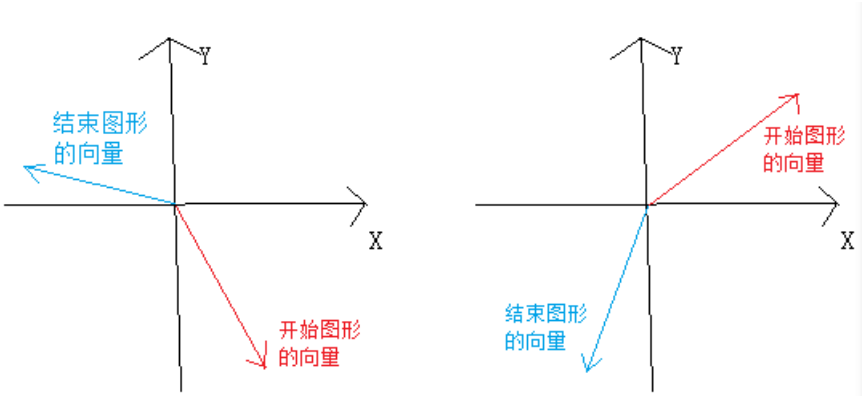
矢量线性插值，是在给定初始点和终止点集合后，将 n 个点转换为 n-1 个按顺序首尾相连的向量。然后，再将向量转化为极坐标，并对长度和角度进行线性插值。相当于将其转换到另外一个空间，维护了其角度和长度上的连续性。其过程中的极坐标长度和角度的计算公式如下：

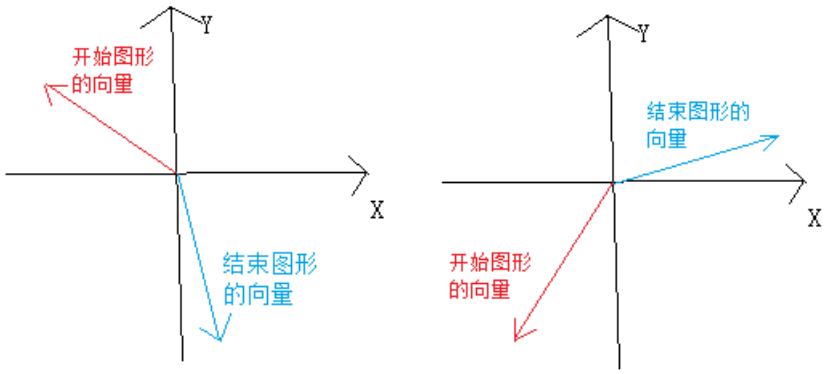
$$r = r0*t + r1*(1-t);$$

$$a = a0*t + a1*(1-t);$$

其中有一些角度会有开始图形的向量与结束图形的向量的夹角中有一个大于 180° 的情况，我们选择小于 180° 的方向进行旋转。

（如下所示的四种情况）





相关的代码如下：

```
1.  float v = (float)this->time / this->number;
2.  this->process_shape.addPoint((1-v)*this->begin_shape.point[0] + v*this->end_
    shape.point[0]); //第一个点用线性插值算法
3.  for(int i=1; i<this->begin_shape.point.size(); i++){
4.
5.      //开始图形的向量
6.      QVector2D d0 = QVector2D(this->begin_shape.point[i]-this->begin_shape.po
    int[i-1]);
7.      float r0 = d0.length();
8.      float a0 = qAtan(d0.y()/d0.x());
9.      if(d0.x()<0) a0 += pi;
10.
11.     //结束图形的向量
12.     QVector2D d1 = QVector2D(this->end_shape.point[i]-this->end_shape.point[
    i-1]);
13.     float r1 = d1.length();
14.     float a1 = qAtan(d1.y()/d1.x());
15.     if(d1.x()<0) a1 += pi;
16.
17.     //过程向量
18.     float ri = (1-v)*r0 + v*r1;
19.     float ai = (1-v)*a0 + v*a1;
20.
21.     //当两个向量的夹角中有一个大于 180°时，选择小于 180°的方向进行旋转
22.     if((a0 <= 0) && (a1 >= a0 + pi)) ai = (1-v)*a0 + v*a1 - 2*pi*v;
23.     if((0 <= a0) && (a0 <= ai/2)  && (a0 + pi <= a1 )) ai = (1-v)*a0 + v*a1
    - 2*pi*v;
24.     if((pi/2 <= a0) && (a0 <= pi) && (a1 <= a0 - pi)) ai = (1-v)*a0 + v*a1 +
     2*pi*v;
25.     if((pi <= a0) && (a0 <= 3*pi/2) && (a1 <= a0 - pi)) ai = (1-v)*a0 + v*a1
     + 2*pi*v;
```

```
26.
27.     QVector2D di = QVector2D(ri*qCos(ai),ri*qSin(ai));
28.     this->process_shape.addPoint(this->process_shape.point[i-1]+di.toPoint()
    );
29. }
```

（3）在变化过程中速度的选择。

在变化过程中有三种速度的选择：匀速，加速或减速，其公式如下：

等速: law=t;

加速: law=(1-cos(PI*t/2));

减速: law=sin(PI*t/2);

在程序中，速度用 v 来表示，相关代码如下：

```
1.  float v = (float)this->time / this->number;
2.  if(this->speed == 1) v = 1 - qCos(pi/2*v); //加速
3.  if(this->speed == 2) v = qSin(pi/2*v); //减速
```

2. 已绘制图形的存储及操作

（1）Shape 类

```
1.  class Shape
2.  {
3.  public:
4.      QVector<QPoint> point;
5.      Shape();
6.      void addPoint(QPoint point); //添加点
7.      void undo(); //撤回点的绘制
8.      void clear(); //清空
9.  };
```

（2）相关函数的操作

①void Shape::**addPoint**(QPoint point)函数，用于添加点

```
1.  void Shape::addPoint(QPoint point)
2.  {
3.      this->point.append(point);
4.  }
```

②void Shape::**undo**()函数，用于撤回点的绘制

```
1.  //撤回点的绘制
2.  void Shape::undo()
3.  {
4.      this->point.pop_back();
5.  }
```

③void Shape::**clear**()函数，用于清空所有点

```
1.  //清空
2.  void Shape::clear()
3.  {
4.      this->point.clear();
5.  }
```

3. 建立可视化窗口，通过鼠标点击绘制开始图形和结束图形。

（1） 窗口的选择

QT 有三大窗口，分别是 QWidget，QMainWindow 和 QDialog。其中 QWidget 类是所有用户界面对象的基类。 窗口部件是用户界面的一个基本单元：它从窗口系统接收鼠标、键盘和其它事件，并且在屏幕上绘制自己；QMainWindow 类提供一个有菜单条、锚接窗口（例如工具条）和一个状态条的主应用程序窗口。主窗口通常用在提供一个大的中央窗口部件（例如文本编辑或者绘制画布）以及周围菜单、工具条和一个状态条。QMainWindow 常常被继承，因为这使得封装中央部件、菜单和工具条以及窗口状态条变得更容易，当用户点击菜单项或者工具条按钮时，槽会被调用；QDialog 是最普通的顶级窗口，通常情况下，顶级窗口部件是有框架和标题栏的窗口。

由于在本次实验中有涉及到鼠标有关的事件，所以我创建 Widget 类作为应用程序的主界面。

对于窗口的设置：

```
1.  //设置画布
2.  paint = new QPainter;
3.  paint->begin(this);
4.  paint->drawRect(0, 0, 650, 540); //画布大小
```

（2） 与绘制开始图形和结束图形有关的成员变量：

```
QPainter *paint;
QPoint point;
Shape begin_shape; //开始的图形
Shape end_shape; //结束时的图形
Shape process_shape; //从开始到结束过程中变化的图形
bool isClicked; //Begin 按钮是否被点击
bool isBegin; //绘制的图形是否为开始的图形
```

其中鼠标点击的点会被临时存储在 point 中，begin_shape 代表开始图形，end_shape 代表结束图形，process_shape 代表从开始到结束过程中变化的图形，isClicked 代表 Begin 按钮是否被点击，代表绘制是否开始，isBegin 代表绘制的图形是开始图形还是结束图形。。

（3） 与绘制开始图形和结束图形有关的函数：

①函数 void Widget::***mousePressEvent***(QMouseEvent *e)，首先需要判断是否已经开始绘制图形（即是否已经点击 Begin 按钮），开始绘制后记录鼠标所点击的位置（即点的横纵坐标），并对该点进行操作和编辑，并在函数最后进行 update，在动画期间绝大多数情况下，update 允许 Qt 来优化速度并且防止闪烁，，当 Qt 回到主事件中时，它规划了所要处理的绘制事件。这样允许 Qt 进行优化，从而得到更快的速度和更少的闪烁。

相关代码如下：

```
void Widget::mousePressEvent(QMouseEvent *e)
{
    if(isClicked == true){
        this->point = e->pos();
        this->editPoint();
        update();
    }
}
```

其中的应用的 editPoint()函数对鼠标绘制的点进行编辑，进行判断将其加入对应的图形中去，定义如下：

```
void Widget::editPoint()
{
    if(this->point.x() > 0 && this->point.x() < 650 && this->point.y() > 0 &
& this->point.y() < 540){
        if(this->isBegin == true) //绘制的是开始的图形
            this->begin_shape.addPoint(this->point);
```

```
        if(this->isBegin == false) //绘制的是结束的图形
            this->end_shape.addPoint(this->point);
    }
}
```

②函数 void Widget::***paintEvent***(QPaintEvent *)，用来绘制开始图形、结束图形及结束图形。其函数思路为根据开始图形、结束图形、中间图形的点分别绘制开始图形、结束图形和中间图形，在该函数中，绘制直线时运用的是 void drawLine()函数，绘制点时运用的是 void drawEllipse()函数，通过调整半径的大小可以调整点的大小，比较明显，而在上次作业中所查到的 void drawPoint()函数所呈现出来的点并不明显，所以在这里依然运用了 void drawEllipse()函数；在该函数中，绘制开始图形和结束图形直线所用的颜色为深蓝色，绘制其点用的颜色为粉红色；绘制过程中图形直线所用的颜色为绿色，绘制其点所用的颜色为蓝色。

与该部分相关的代码如下：

```
void Widget::paintEvent(QPaintEvent *)
{

    paint = new QPainter;
    paint->begin(this);

    //绘制开始图形
    for(int i = 0; i < this->begin_shape.point.size(); i++){
        paint->setPen(QPen(QColor(240,100,100),2));
        paint->drawEllipse(this->begin_shape.point[i],2,2);
        if(i>=1){
            paint->setPen(QPen(QColor(50,50,180),2));
            paint->drawLine(this->begin_shape.point[i-1],this->begin_shape.point[i]);
        }
    }

    //绘制结束图形
    for(int i = 0; i < this->end_shape.point.size(); i++){
        paint->setPen(QPen(QColor(240,100,100),2));
        paint->drawEllipse(this->end_shape.point[i],2,2);
        if(i>=1){
            paint->setPen(QPen(QColor(50,50,180),2));
            paint->drawLine(this->end_shape.point[i-1],this->end_shape.point[i]);
```

```
        }
    }

    //绘制过程中的图形
    for(int i = 0; i < this->process_shape.point.size(); i++){
        paint->setPen(QPen(QColor(0,255,255),2));
        paint->drawEllipse(this->process_shape.point[i],2,2);
        if(i>=1){
            paint->setPen(QPen(QColor(100,200,20),2));
            paint->drawLine(this->process_shape.point[i-1],this->process_sha
pe.point[i]);
        }
    }

    paint->~QPainter();

}
```

③函数 void Widget::**on_begin_pushbutton_clicked**()，用来代表绘制图形的开始，同时激活 end 按钮和 undo 按钮。

与该部分相关的代码如下：

```
void Widget::on_begin_pushbutton_clicked()
{
    isClicked = true;
    //设置按钮状态
    this->ui->end_pushbutton->setEnabled("true");
    this->ui->undo_pushbutton->setEnabled("true");
}
```

④函数 void Widget::**on_undo_pushbutton_clicked**()，用来撤销开始图形或者结束图形的绘制，即返回上一步。当撤销按钮被按下时，判断此步撤销的是开始图形还是结束图形，在进行相应的操作，最后进行 update().

与此相关的代码如下：

```
void Widget::on_undo_pushbutton_clicked()
{
    //判断此步撤销的是开始图形还是结束图形
    if(this->isBegin == true) this->begin_shape.undo();
    else this->end_shape.undo();
    update();
```

```
}
```

⑤函数 void Widget::**on_end_pushbutton_clicked**()，用来表示开始图形或者结束图形的绘制结束。当结束按钮被按下时，判断开始图形是否已经完成绘制，如果未开始绘制，则继续开始绘制开始图形；若开始图形已经完成绘制，则进行判断结束图形是否已经完成绘制，如果结束图形未开始绘制，则绘制第二次图形，如果结束图形已经绘制，则设置不允许继续绘制，并激活 start 按钮和 stop 按钮，可以开始进行动画演示。

与该部分有关的函数如下：

```
void Widget::on_end_pushbutton_clicked()
{
    if(this->begin_shape.point.isEmpty()){ //开始的图形尚未绘制
        this->isBegin = true;
        this->ui->undo_pushbutton->setEnabled("true");
        this->ui->begin_pushbutton->setDisabled("true");
        this->ui->end_pushbutton->setText("End1");
    }
    else if(!this->begin_shape.point.isEmpty()){ //开始的图形绘制完毕
        this->isBegin = false;
        if(this->end_shape.point.isEmpty()){ //结束的图形尚未绘制
            //设置按钮状态
            this->ui->undo_pushbutton->setEnabled("true");
            this->ui->begin_pushbutton->setDisabled("true");
            this->ui->end_pushbutton->setText("End2");
        }
        if(!this->end_shape.point.isEmpty()){ //结束的图形绘制完毕
            //设置按钮状态
            this->ui->undo_pushbutton->setDisabled("true");
            this->ui->begin_pushbutton->setDisabled("true");
            this->ui->stop_pushbutton->setEnabled("true");
            this->ui->start_pushbutton->setEnabled("true");
            this->ui->end_pushbutton->setText("Done");
            isClicked = false;
        }
    }
}
```

⑥函数 void Widget::**on_clear_pushbutton_clicked**()用来清除画布进行下一次绘制，将有关变量清空。与此相关的代码如下

```
void Widget::on_clear_pushbutton_clicked()
```

```
{
    //设置画布
    paint = new QPainter;
    paint->begin(this);
    paint->drawRect(0, 0, 650, 540); //画布大小

    this->time = 0;
    this->total_time = 0;
    this->number = 0;
    this->speed = 0;

    //清空已有的图形
    this->begin_shape.clear();
    this->process_shape.clear();
    this->end_shape.clear();
    this->isBegin = true;


    //设置按钮状态
    isLinear = true;
    this->ui->end_pushbutton->setText("End1");
    this->ui->begin_pushbutton->setEnabled("true");
    this->ui->undo_pushbutton->setDisabled("true");
    this->ui->end_pushbutton->setDisabled("true");
    this->ui->stop_pushbutton->setDisabled("true");
    this->ui->start_pushbutton->setDisabled("true");
    timer = new QTimer(this);
    connect(timer,SIGNAL(timeout()),this,SLOT(animation())); //当达到超过时间，则发射信号，执行指定的槽函数
    isStart = false;
    isClicked = false;

    update();
}
```

4. 控制变形过程中的图形变化。

（1）与变形有关的成员变量

```
bool isLinear; //变化模式为线性插值或线性矢量插值
QTimer *timer;
int speed; //速度
int time;
int total_time;
```

```
int number;
bool isStart; //是否开始演示变化过程
```

其中 is_Linear 来表示变化模式为线性插值或线性矢量插值，speed 用来表示所选用的速度模式，time 用来表示在演示过程中所花费的时间，total_time 用来读取 ui 界面上用户选择的时间长短，number 用来表示之后用来循环的时间。

（2）与时间及绘制动画有关的设置。

首先将时间信号和槽通过 connect 建立连接，设置初始变化模式为线性插值，设置初始播放状态为不播放。

与该部分有关的代码如下：

```
this->time = 0;
timer = new QTimer(this);
//信号和槽通过 connect 建立连接
connect(timer,SIGNAL(timeout()),this,SLOT(animation())); //当达到超过时间，则发射信号，执行指定的槽函数
this->isLinear = true; //设置初始变化模式为线性插值
isStart = false; //设置初始播放状态为不播放
```

（3）与时间及绘制动画有关的函数

①函数 void Widget::**on_start_pushbutton_clicked**()，当按下”Start”按钮时触发变化过程。首先判断动画是否已经开始，如果动画未开始，则绘制动画，读取选择的速度模式和时间长度，启用定时器，每过一定的时间会变化中间过程中的图形；如果动画已经开始，则重新绘制动画，重新计时。进行 update()。

与该部分有关的代码如下：

```
void Widget::on_start_pushbutton_clicked()
{
    //判断动画是否已经开始：未开始，绘制动画
    if(isStart == false){
        if(this->ui->linear_radiobutton->isChecked()){
            isLinear = true;
        }
        if(this->ui->vector_radiobutton->isChecked()){
            isLinear = false;
        }

        this->speed = this->ui->speed_box->currentIndex(); //速度模式
        this->total_time = this->ui->time_box->value();
        this->number = this->total_time * 50;
        timer->start(5);
        isStart = true;
    }
```

```
18.
19.     //判断动画是否已经开始：已经开始，则重新绘制动画
20.     else{
21.         isStart = false;
22.         this->time = 0;
23.         if(timer->isActive()) timer->stop();
24.         this->process_shape.clear();
25.         update();
26.     }
27. }
```

②函数 void Widget::**on_stop_pushbutton_clicked**()用来中止变化过程，停止计时。与这部分有关的代码如下：

```
1.  void Widget::on_stop_pushbutton_clicked()
2.  {
3.      isStart = false;
4.      if(timer->isActive()) timer->stop();
5.  }
```

③函数 void Widget::**animation**()用来控制中间过程中的变化。首先判断绘制模式为线性插值还是矢量线性插值，再分别根据两种不同的情况用第一部分的代码进行处理，最后对时间进行修改，当时间结束时停止播放变化过程。

该部分的完整代码如下：

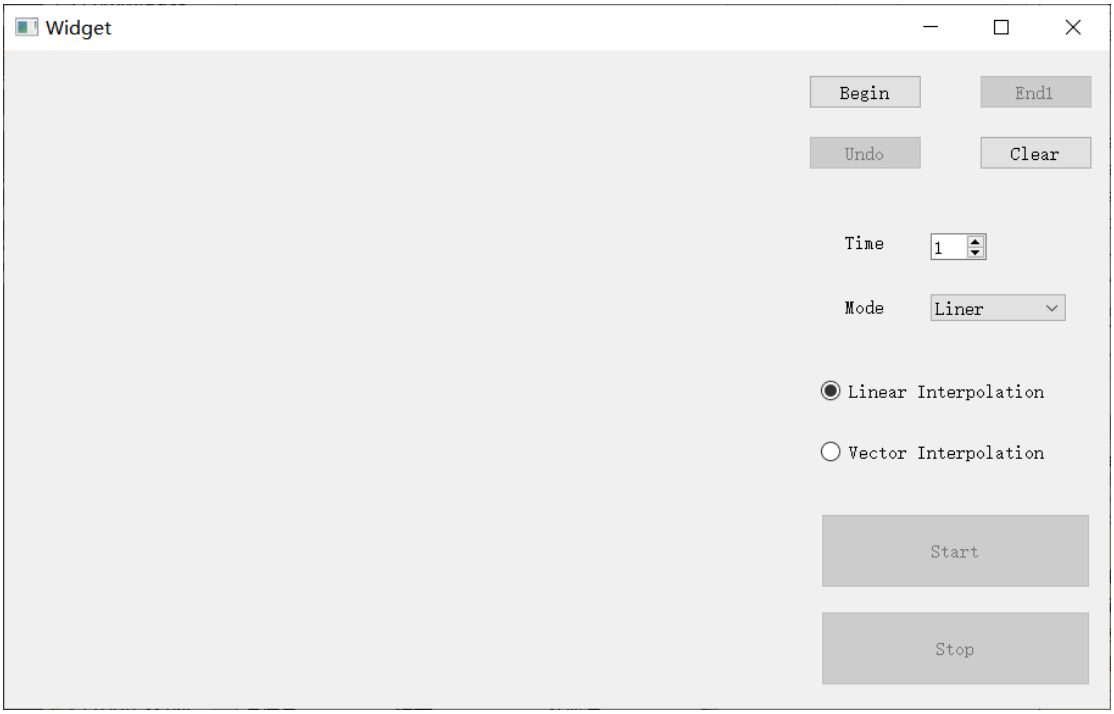
```
1.  void Widget::animation()
2.  {
3.      float pi = 3.14; //Π
4.      if(isLinear == true){ //线性插值绘制模式
5.          this->process_shape.clear();
6.          float v = (float)this->time / this->number;
7.          if(this->speed == 1) v = 1 - qCos(pi/2*v); //加速
8.          if(this->speed == 2) v = qSin(pi/2*v); //减速
9.          for(int i=0; i < this->begin_shape.point.size(); i++){
10.             this->process_shape.addPoint((1-v)*this->begin_shape.point[i] +
    v*this->end_shape.point[i]);
11.         }
12.     }
13.     else{ //矢量线性插值绘制模式
14.         this->process_shape.clear();
15.         float v = (float)this->time / this->number;
16.         if(this->speed == 1) v = 1 - qCos(pi/2*v); //加速
17.         if(this->speed == 2) v = qSin(pi/2*v); //减速
```

```
        this->process_shape.addPoint((1-v)*this->begin_shape.point[0] + v*this->end_shape.point[0]); //第一个点用线性插值算法
        for(int i=1; i<this->begin_shape.point.size(); i++){

            //开始图形的向量
            QVector2D d0 = QVector2D(this->begin_shape.point[i]-this->begin_shape.point[i-1]);
            float r0 = d0.length();
            float a0 = qAtan(d0.y()/d0.x());
            if(d0.x()<0) a0 += pi;

            //结束图形的向量
            QVector2D d1 = QVector2D(this->end_shape.point[i]-this->end_shape.point[i-1]);
            float r1 = d1.length();
            float a1 = qAtan(d1.y()/d1.x());
            if(d1.x()<0) a1 += pi;

            //过程向量
            float ri = (1-v)*r0 + v*r1;
            float ai = (1-v)*a0 + v*a1;

            //当两个向量的夹角中有一个大于 180°时，选择小于 180°的方向进行旋转
            if((a0 <= 0) && (a1 >= a0 + pi)) ai = (1-v)*a0 + v*a1 - 2*pi*v;

            if((0 <= a0) && (a0 <= ai/2)  && (a0 + pi <= a1 )) ai = (1-v)*a0 + v*a1 - 2*pi*v;
            if((pi/2 <= a0) && (a0 <= pi) && (a1 <= a0 - pi)) ai = (1-v)*a0 + v*a1 + 2*pi*v;
            if((pi <= a0) && (a0 <= 3*pi/2) && (a1 <= a0 - pi)) ai = (1-v)*a0 + v*a1 + 2*pi*v;

            QVector2D di = QVector2D(ri*qCos(ai),ri*qSin(ai));
            this->process_shape.addPoint(this->process_shape.point[i-1]+di.toPoint());
        }
    }

    this->time++;
    if(this->time > this->number){ //时间结束时停止播放变化过程
        this->time = 0;
        timer->stop();
    }
    update();
```

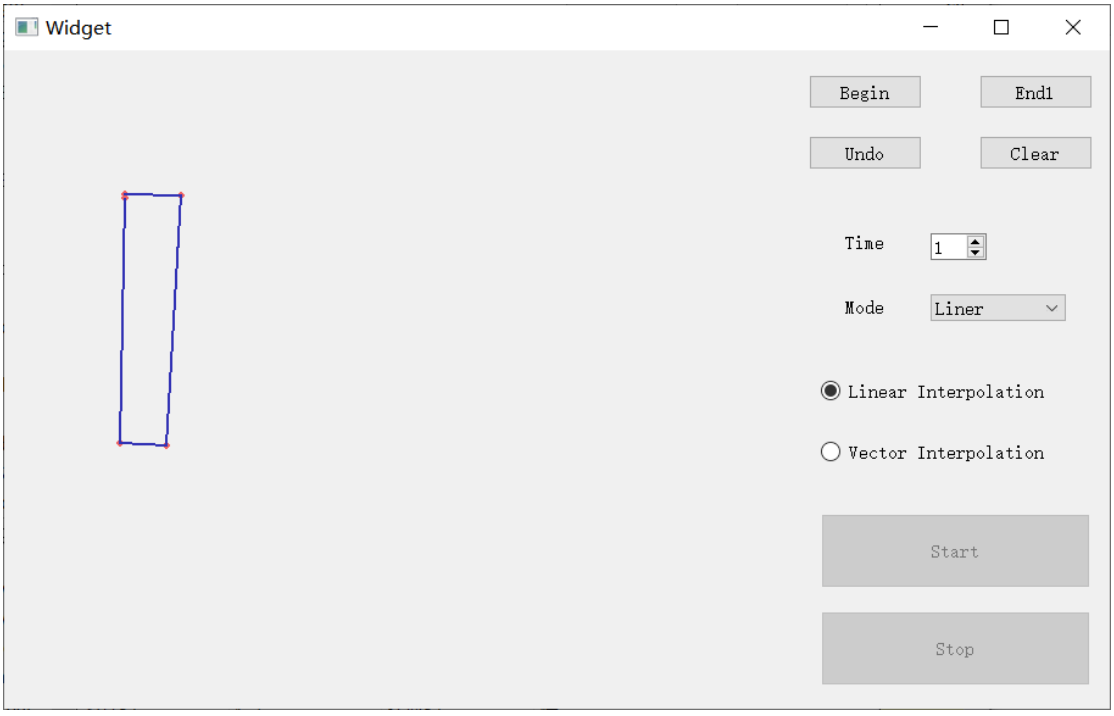
```
54. }
```

五、 实验结果

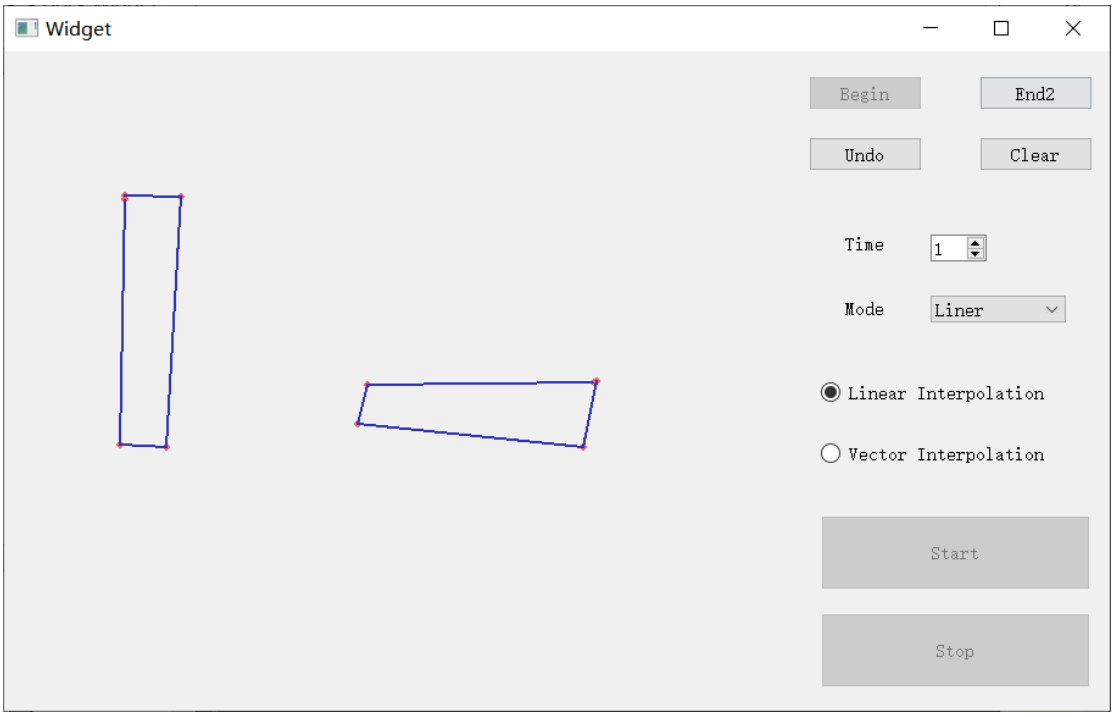
1.初始界面：



2.绘制开始图形

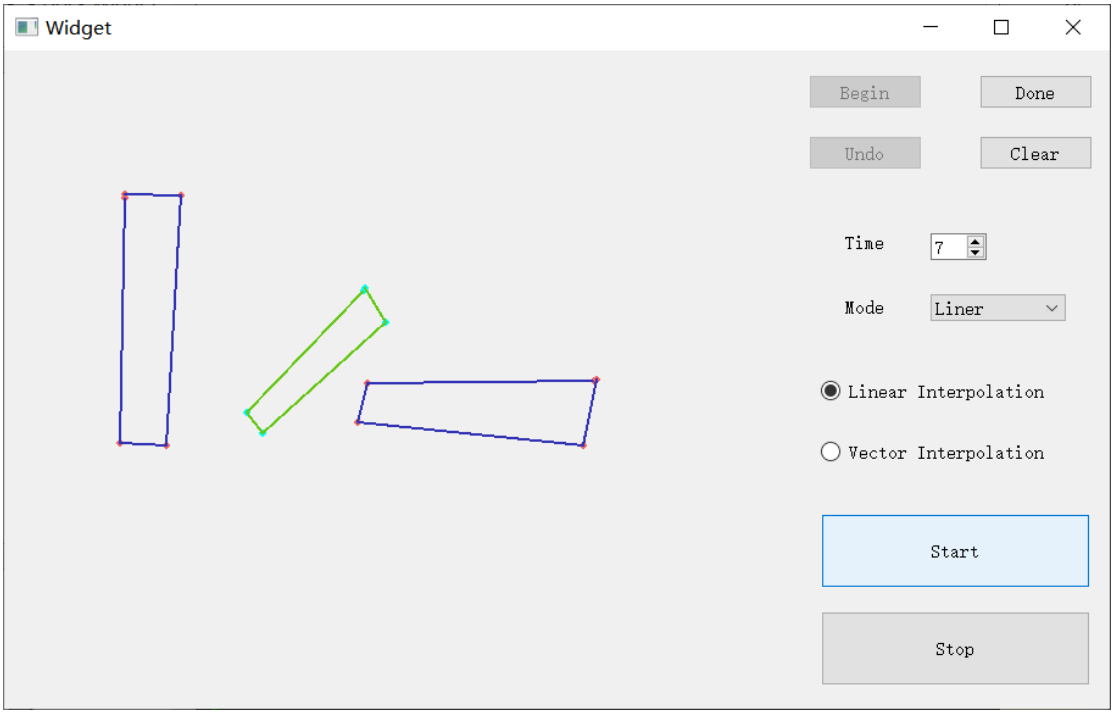


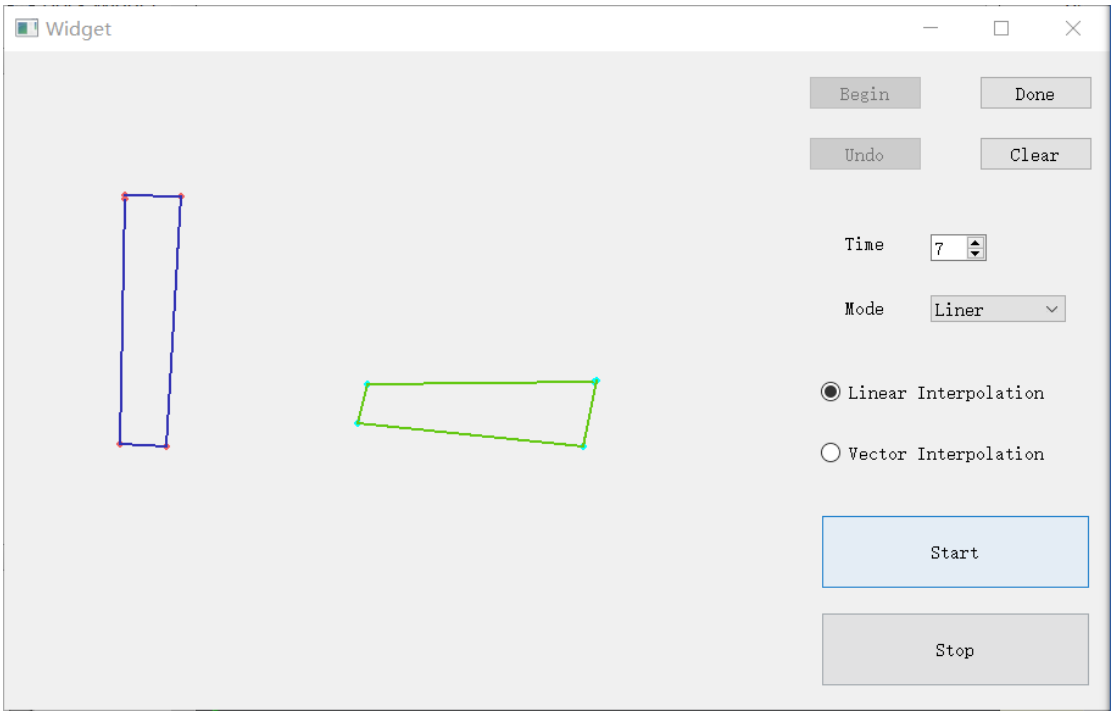
3.绘制结束图形



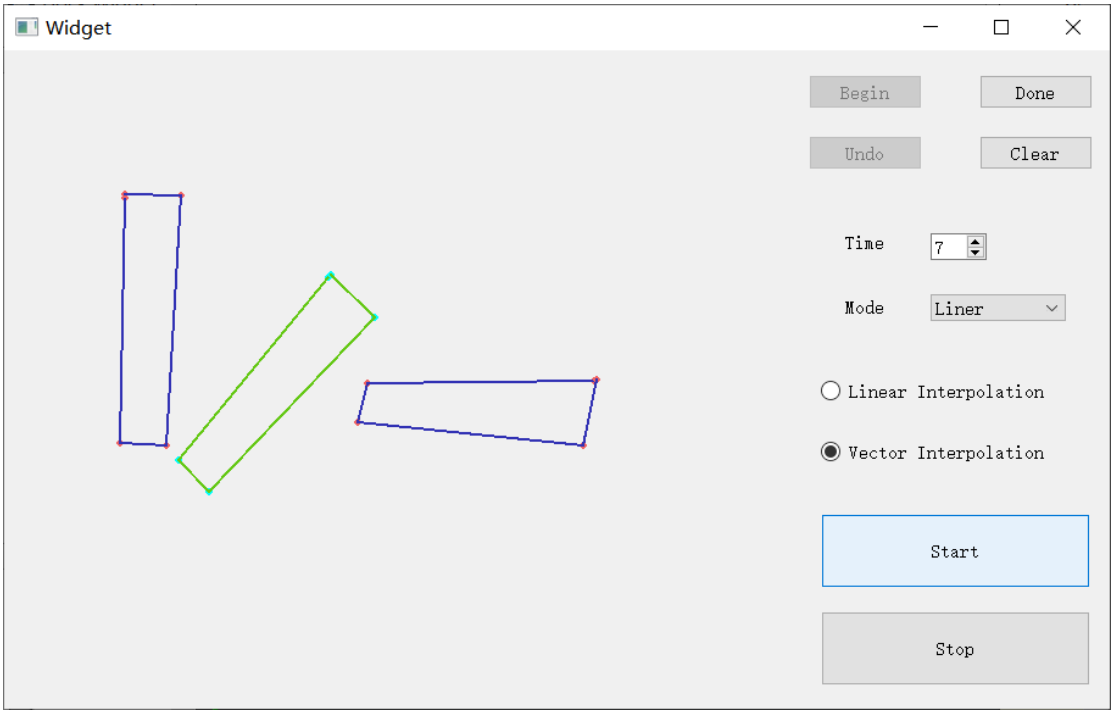
4.变化过程

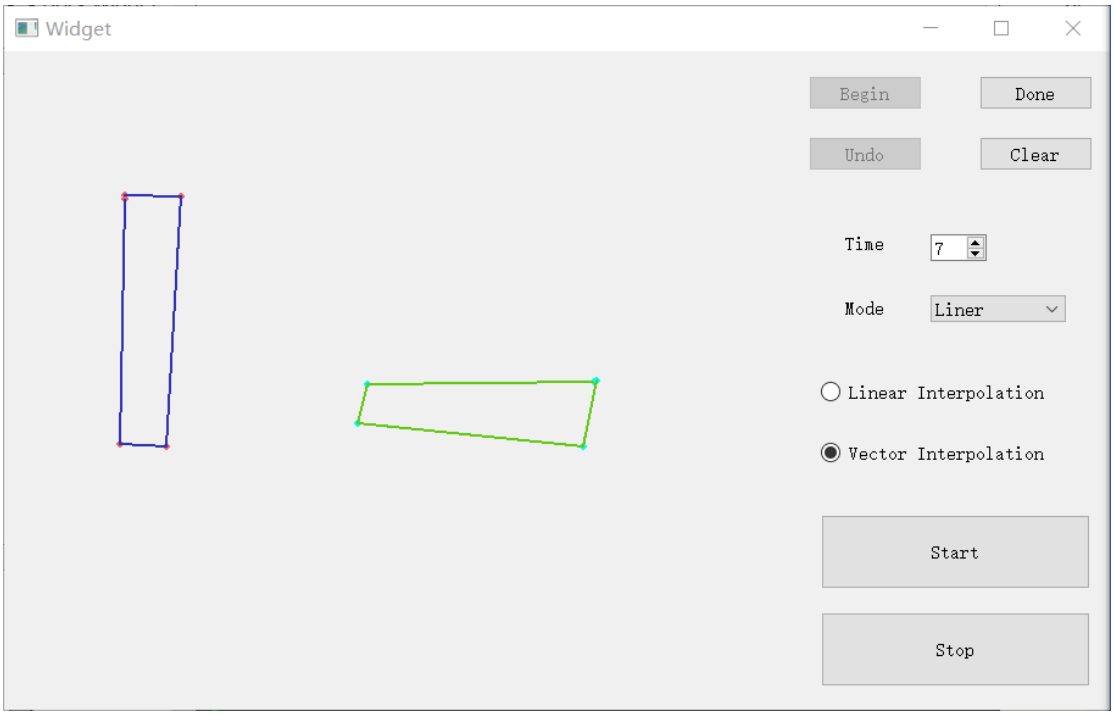
（1）线性插值：




Widget
Begin
Done
Undo
Clear
Time
7
Mode
Liner
Linear Interpolation
Vector Interpolation
Start
Stop

（2）矢量线性插值


Widget
Begin
Done
Undo
Clear
Time
7
Mode
Liner
Linear Interpolation
Vector Interpolation
Start
Stop



5.改变速度

变化速度在截图中难以显示，详见 video.mp4。

由以上结果可以看到，本次实验实现了实验的目的和要求，达到了比较不错的结果。

六、 实验感想及分析

通过本次实验，我掌握了线性插值和矢量线性插值的思路和算法，了解了两种变形算法不同的性能和对变形过程中对于图形形状的影响，掌握并实践了线性插值和矢量线性插值的具体思路和程序代码的对应关系；了解了关键帧动画系统的结构和动画动态控制的基本原理和方法。同时，在完成本次实验的过程中，通过查找资料和 debug，我学会了很多 QT 的函数及其用法，懂得了更多对于 ui 界面的操作，更加熟练。

在本次实验中，我遇到的困难主要在处理变量和运用矢量线性插值绘制变化过程方面。因为本次实验涉及到的需要进行判断的变量非常多，如绘制模式的选择，是否开始绘制，绘制的是否是开始图形，正在绘制的是开始图形还是结束图形，是否可以进行绘制等，在设置过程中经常会出现差错，理清思路后解决了这个问题。在运用矢量矢量线性插值绘制变化过程的过程中，经常会出现绘制过程中变化异常、旋转方向错误造成形状分解的问题，在分析了角度的变化和几种特殊情况并特殊设置后基本解决了这个问题。

通过本次实验，我感觉到自己对于关键帧动画系统的结构和动画动态控制有了更深的理解，受益匪浅。同时，钻研问题和解决问题是很快乐的，而在不断解决问题的过程中我感受到了自己的成长，感觉很有收获。