

WPF 技术在蒙塞尔色相仿真测试中的应用

李晓京,马 进,胡文东,张利利,惠铎铎

(第四军医大学 航空航天医学教育部重点实验室,陕西 西安 710032)

摘 要:蒙塞尔色相测试(Farnsworth-Munsell Hue Test)用于检测色觉缺陷人员的色弱区域,该测试的计算机仿真应用在一些方面存在不足。文中在对蒙塞尔测试需求分析的基础上,提出了基于 .Net WPF 技术的色相子操作仿真软件功能设计方案,重点阐述了色相子控件类、控件拖放行为类、基于属性触发器的层次效果、泛型容器排序算法以及由故事板控制的色相子移位动画设计思路。最终实现的色相子操作仿真界面具有分辨率自适应、色相子操作仿真特效、色相子移位多重业务逻辑等特点,能够充分满足目标应用需求。

关键词:WPF;Farnsworth-Munsell 色相测试;自定义控件;行为类;属性触发器;泛型;故事板

中图分类号:TP311 **文献标识码:**A **文章编号:**1673-629X(2016)02-0154-07

doi:10.3969/j.issn.1673-629X.2016.02.035

Application of WPF in Farnsworth-Munsell Hue Simulation Testing System Development

LI Xiao-jing,MA Jin,HU Wen-dong,ZHANG Li-li,HUI Duo-duo
(Key Laboratory of Aerospace Medical of Ministry of Education,Fourth Military Medical University,
Xi'an 710032,China)

Abstract:Farnsworth-Munsell Hue Test is applied to inspect the color weakness area of hypochromatopsia patients. But,for the computer simulation software of the test,there are some improvable aspects in the characteristics' simulation. Based on the demand analysis of the test,a kind of software function and frame,utilizing .NET WPF,was developed for Hue-Token (HT) manipulating simulation. Primarily,the design of HT component,drag-drop behavior class,layer effect based on property trigger,sorting arithmetic of generic class and HT shifting animation under the control of storyboard were described in detail. Features of the final simulation interface such as resolution self-adaption,special effects of HT operation,multi-business logic of HT shift,could meet the need of demand of object application sufficiently.

Key words:WPF (Windows Presentation Foundation);Farnsworth-Munsell Hue Simulation Testing;user control;behavior class;property trigger;generic type;storyboard

0 引 言

WPF(Windows Presentation Foundation)是一个用于 Windows 的针对 .NET 设计的图形显示系统,基于 WPF 开发的应用程序界面底层都使用 DirectX 技术,从而使得再普通不过的应用程序也能使用诸如半透明、抗锯齿等丰富的效果,并由 DirectX 将图形渲染任务尽可能多地交给显卡 GPU 处理,从而大大提高了硬件加速性能。其“界面 UI 设计”与“后台编程”相分离的设计思想,使得界面元素外观能够以类似图形编辑的方式轻松实现,使得编程人员得以专注于代码和业

务逻辑分析,二相结合创造出优秀用户体验的交互设计。另外,WPF 可以实现资源构件化开发,资源的设计与开发具有很高的可重用性^[1-9]。

蒙塞尔色相测试(Farnsworth-Munsell Hue Test)用于检测色觉缺陷人员的色弱区域^[10]。目前可以找到一些 Win32 仿真测试软件^[11-13],在实用时发现存在一些不足:

(1)选中的色相子缺乏相对其他色相子的空间层次效果;

(2)拖拽释放色相子对象后,色相子队列的重排

是一个跳变过程,色相子对象位移没有过渡,易使受测者产生困惑和怀疑;

(3)软件界面大小无法调节,在高分辨率显示器上不能全屏会显得交互界面过小。

1 蒙塞尔色相子仿真软件需求分析

从测试实物外观及相应仿真软件的不足之处可见,该仿真应用存在如下关键需求:

色相子仿真控件:由于实物色相子具有相同的外观,仅色相不同且数量较多,因此合理的作法是将其设计为可重用控件,提供颜色设置属性以便编程。

色相子拖拽跟随及层次特效:采用鼠标操作模拟手工挪动色相子的位置变化,同时添加阴影效果,模拟控件的悬空状态。

色相子释放后的排序业务逻辑:在色相子拖拽释放后,需要根据其新的位置信息来判断它应该归置于什么标准位置,从而引起整个色相子队列的重排。

色相子归位动画:用户在释放色相子控件后,如果参与排序业务的色相子控件都直接按照排序结果直接出现在其标准位置上,无疑是一种位置和色彩的突变,在视觉上会给用户带来不适以及对排序结果的怀疑与错愕。因此,在完成排序业务逻辑运算后,让色相子控件以过渡动画的形式运动到它们的标准位置上,一方面与现实情况更为接近,另一方面可以让用户直观地观察到排序动态过程,从而增强其操作信心。

分辨率无关的自动缩放效果:由于用户显示器尺寸和分辨率设置有种种可能,目标应用应能随着用户控制交互界面的放大、缩小而自适应地调整界面控件元素的大小,从而满足不同用户的喜好。

从前述可知,WPF 技术可以全方位满足目标应用需求,特别是在用户界面的美观优化与交互人性化设计方面,具有显著的优势。

2 色相子控件设计

在 WPF 丰富的绘图模型以及声明式用户界面的支持下,与专业设计工具 Expression Blend 相配合,图形自定义控件的开发变得简单而直观。在 Expression Blend 中导入丰富的图形构建更加漂亮的控件,与在 Photoshop 中进行图形处理同样容易^[14]。简便起见,下面以一个简单明了的自定义控件加以演示和说明。

自定义控件的可视化树及外观如图 1 所示。

由可视化树可见,派生于 UserControl 的色相子控件由简单的四个元素构成。其中,第一个 path 元素是一个无色圆形,将其填充画刷命名为 CColor;第二个 path 则仿真色相子的环状边框覆盖在上方。这两个 path 元素嵌套在 Canvas 面板中置于 Viewbox 元素中,

通过将 Viewbox 元素的 Stretch 属性设置为"Uniform"实现控件在应用时的一致性缩放。

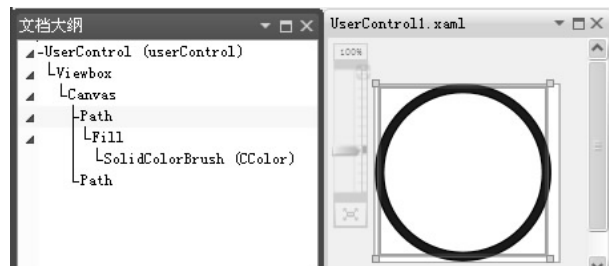


图 1 色相子控件可视化外观

后台代码工作主要是为控件创建三个依赖项属性:ToTop、ToLeft、IsPressed。前两个属性用于控件实例的拖放归位动画,后者用于控件实例的拖放状态层次效果属性触发器。将前述 CColor 画刷的 Color 属性封装为控件的 CenterColor 属性公开,通过该属性可直接设定色相子控件实例的中心颜色。

主要代码如下:

```
public partial class UserControl : UserControl
{
    public Color CenterColor //该属性用于设置色相子的颜色
    {
        set { CColor.Color = value; }
        get { return (Color)CColor.Color; }
    }

    // ToLeftProperty 依赖项属性用于控件拖放归位动画
    public static readonly DependencyProperty ToLeftProperty =
        DependencyProperty.Register("ToLeft",
            typeof(double),
            typeof(UserControl1),
            new FrameworkPropertyMetadata(0.0, null));
    public double ToLeft
    {
        set { SetValue(ToLeftProperty, value); }
        get { return (double)GetValue(ToLeftProperty); }
    }

    // ToTopProperty 依赖项属性与 ToLeftProperty 类似,定义从略

    // IsPressedProperty 依赖项属性用于触发控件拖放层次效果, bool 型, 定义从略
    .....
    public UserControl1()
    {
        InitializeComponent();
    }
}
```

需要强调的是,依赖项属性的性质决定了它支持动态绑定,因此尤其适用于动画、界面元素动态更新等需求,从而极大地简化了编程复杂度,但相应地也会为其强大动态功能付出系统资源代价。因此,在普通属

性和依赖性属性的选择方面要因地制宜。色相子控件中,CenterColor 属性就被声明为普通属性,这是由于色相子的中心颜色在程序生命周期中不会有动态的变化,因此普通属性就可以满足需要。此外,并未为CenterColor 属性创建相应的私有数据,而是通过访问函数直接对其他元素属性进行操作,这种方式巧妙地

达到了数据封装和降低系统资源占用的目的,是 .Net 程序设计中一种非常实用的技巧。

3 色相子交互界面设计

仿真色相子界面用户交互流程见图 2。

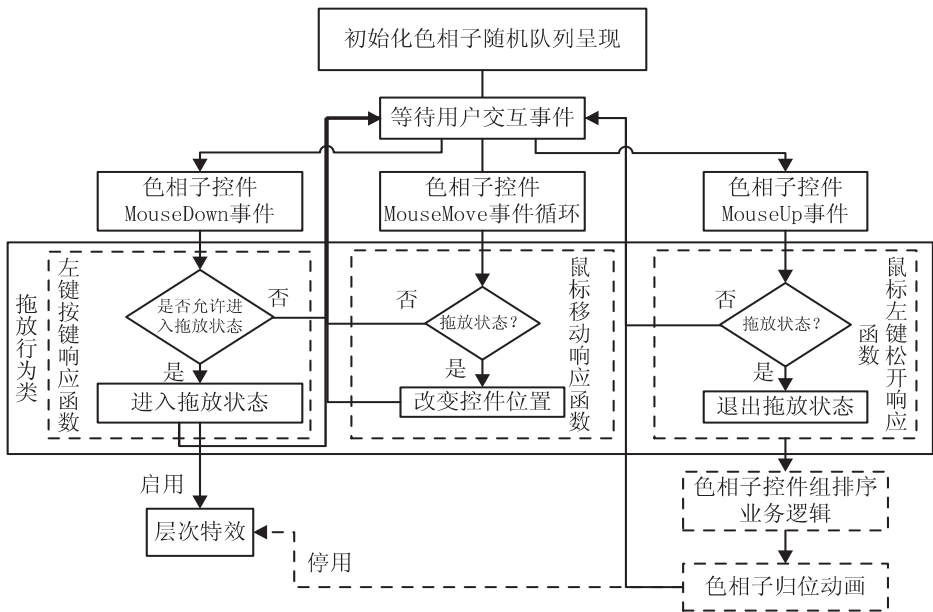


图 2 仿真色相子控件用户交互流程

可见,由色相子控件事件驱动的拖放行为类、层次特效、控件排序业务逻辑、色相子归位动画等功能模块相互作用,构成了色相子仿真界面的人机交互逻辑。

3.1 色相子拖放行为类

WPF 行为特征旨在代码重用,封装好的行为类可以为用户界面上具有相同操作响应的元素提供统一的操作支持。设计拖放行为类可以满足色相子的拖放操作仿真,并降低代码工作量。

DragInCanvasBehavior 由泛型类 Behavior<T>派生,其中定义按键、松键等事件响应函数。重载基类的 OnAttached(为元素声明添加行为特征时调用,通常在前台 Xaml 文件声明中为控件启用行为,即附着)、OnDetaching 函数(元素销毁时被调用,即解除附着),在这两个函数中为被附着元素的相应事件委托添加(或删除)相应的行为响应函数。

一个完整色相子的拖放行为仿真由鼠标左键按下、鼠标移动、鼠标左键松开这一系列鼠标事件构成,因此,目标行为类定义了对这三个事件的响应函数。此外,由于拖放操作完成后有后续的排序归位动画,且在动画执行期间不允许执行下一次操作任务,因此,还需要为行为类添加两个委托事件,PreDrag 用于检测是否处于动画状态,DragEnd 用于通知主程序拖放完成可以开始排序归位动画。该类的主要代码如下:

```
{
private Canvas canvas; //用于保存控件的父容器面板
private bool isDragging=false; //用于鼠标移动时判断是否处于拖放状态

private Point mouseOffset; //鼠标位置偏移量
//下两行定义拖放结束事件委托
public delegate void DragEndHandler( Object sender );
public event DragEndHandler DragEnd;
//下两行定义拖放预检事件委托
public delegate bool PreDragHandler( Object sender );
public event PreDragHandler PreDrag;
protected override void OnAttached( )
{ //重载附着函数,将相关响应函数添加到控件委托
base. OnAttached( );
//MouseLeftButtonDown 委托关联,MouseMove、MouseLeftButtonUp 相类从略
this. AssociatedObject. MouseLeftButtonDown += AssociatedObject_MouseLeftButtonDown;
.....
}
protected override void OnDetaching( )
{ //重载解除附着函数,从事件委托列表中删除相关函数
base. OnDetaching( );
//MouseLeftButtonDown 解除委托关联,MouseMove、MouseLeftButtonUp 相类从略
this. AssociatedObject. MouseLeftButtonDown -= AssociatedOb-
```

```
public class DragInCanvasBehavior:Behavior<UIElement>
```

```
ject_MouseLeftButtonDown;

.....

}

private void AssociatedObject_MouseLeftButtonDown ( Object
sender, MouseButtonEventArgs e) //控件鼠标左击事件响应函数
{
    if( null! = PreDrag ) if ( ! PreDrag ( AssociatedObject )) re-
turn; //若不可进入拖放状态,返回
    if( canvas == null )
        canvas = ( Canvas ) VisualTreeHelper. GetParent ( this. Associat-
edObject ); //获取控件父面板
    isDragging = true; //进入拖放状态标志置位
    //获取点击点相对控件的坐标(元素左上为 0,0)
    mouseOffset = e. GetPosition ( AssociatedObject );
    //捕获鼠标,持续接收 mousemove 事件,即使其离开控件
    AssociatedObject. CaptureMouse ( );
}

private void AssociatedObject_MouseMove ( Object sender,
MouseEventArgs e)
{
    //控件鼠标移动事件响应函数
    if( isDragging ) //拖放状态下,控件进入鼠标跟随效果
    {
        //计算控件的新位置并设置其位置附加属性
        Point point = e. GetPosition ( canvas );
        Point pTemp = new Point ( point. X - mouseOffset. X, point. Y -
mouseOffset. Y );
        .....
        //控件移动(应限制控件位置不得超出父面板边界,代码从
        略)
        AssociatedObject. SetValue ( Canvas. TopProperty, pTemp. Y );
        AssociatedObject. SetValue ( Canvas. LeftProperty, pTemp. X );
    }
}

private void AssociatedObject_MouseLeftButtonUp ( Object
sender, MouseButtonEventArgs e)
{
    //控件左键松开响应函数
    if( isDragging )
    {
        AssociatedObject. ReleaseMouseCapture ( ); //解除捕获鼠标
        isDragging = false; //拖放状态标志清除
        //调用拖放结束委托函数,通知主线程执行排序运算和归
        位动画
        if( null! = DragEnd ) DragEnd ( AssociatedObject );
    }
}
```

3.2 色相子控件应用与层次特效

在 Xmal 声明文件中,采用常规的程序集引用和控件声明即可实现色相子控件的应用,以两个 Canvas 面板各含 5 个色相子控件为例,界面元素及可视化树见图 3。

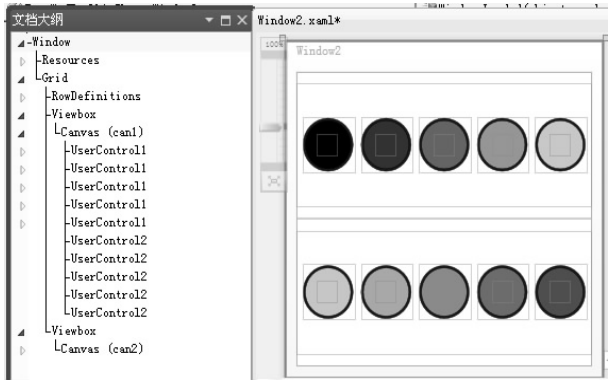


图 3 色相子仿真界面元素及可视化树

注: UserControl2 是用于表示色相子标准位置的图形控件,可忽略。

色相子及行为声明示例代码如下所示:

```
<local:UserControl1 Canvas. Left = "226" Canvas. Top = "30"
CenterColor = "#FF873787" Height = "50" Width = "50" Panel.
ZIndex = "1">
<ii:Interaction. Behaviors>
<local:DragInCanvasBehavior PreDrag = " PreDrag " DragEnd
= " DragFinished" />
</ii:Interaction. Behaviors>
</local:UserControl1>
```

注意在以上声明中,已将后台定义的 PreDrag、DragFinished 函数关联到附着行为类相应委托事件上。

拖放色相子时的层次特效首先是提升色相子在面板中的叠放层次,使受拖放控件呈现于其他控件之上,这将由后台代码来完成。其次,当进入拖动状态时,可创建阴影效果,形成该色相子悬浮于空中的视觉效果。特效在 Xaml 文件的窗体资源中声明,由前述色相子控件的自定义的 IsPressed 属性触发,Xaml 声明如下:

```
<Window. Resources>
<ResourceDictionary>
<Style TargetType = "local:UserControl1">
<Style. Triggers>
<Trigger Property = " local:UserControl1. IsPressed" Value = "
true">
<Setter Property = " local:UserControl1. Effect">
<Setter. Value>
<DropShadowEffect ShadowDepth = "5" Direction = "330"
Color = "#FF313030" Opacity = "0.5" BlurRadius = "5"/>
</Setter. Value>
</Setter>
</Trigger>
</Style. Triggers>
</Style>
</ResourceDictionary>
</Window. Resources>
```

还可以通过故事板为阴影特效创建渐入渐出动画以实现更为逼真的操作效果,这里就不作介绍了。

3.3 色相子排序业务逻辑与归位动画

色相子拖放与归位动画的中间环节是根据其被释放时的位置计算整个色相子队列成员的新位置。为了充分满足用户需求,设计插入和交换两种业务逻辑供用户选择,插入/交换的目标位置可就近或空位侧优先,如图 4 所示。

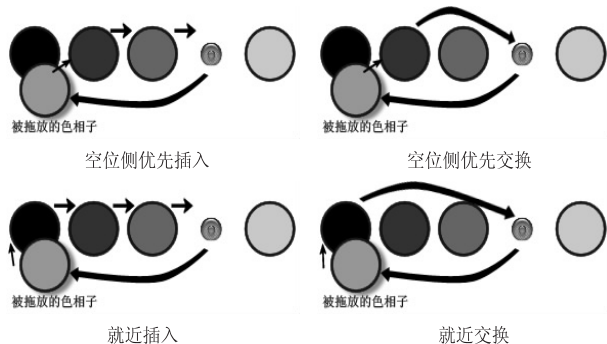


图 4 色相子排序业务逻辑示例

这里以相对复杂的插入逻辑加以说明。

初始化时,将所有色相子控件实例保存在列表泛型容器中,在拖放行为结束时,根据被拖放的色相子位置,调整列表中的控件排序,按排好的顺序将标准位置赋给各色相子的 ToTop、ToLeft 依赖项属性,并进入下一步的归位动画。

归位动画由动态创建的 Storyboard 实例控制,动画开始时将指示动画状态标志量置位,提交给拖放行为类的 PreDrag 委托函数将检测该标志以确定能否进入后续的拖放操作状态。

核心代码如下:

```
public partial class Window2:Window
{
    //多组色相子放在不同的 Canvas 面板中,对每个面板同时只允许执行一个归位动画,因此定义面板与故事板的配对查询字典
    private Dictionary<Canvas, Storyboard> storyboards = new Dictionary<Canvas, Storyboard>();
    //每个面板中所包含的色相子控件组保存在不同的泛型列表容器中,因此定义面板与列表容器的配对查询字典
    private Dictionary<Canvas, List<UserControl1>> dicCanPairedUCList = new Dictionary<Canvas, List<UserControl1>>();
    //示例用两组面板和色相子控件组,因此定义两组保存色相子控件组的列表容器
    List<UserControl1> lstCtr = new List<UserControl1>();
    List<UserControl1> lstCtr2 = new List<UserControl1>();
    //由于色相子 Top 都一样,只须保存各控件 Left 属性值;两组控件共享一组 Left 坐标数组
    List<double> lstXPos = new List<double>();
    public bool bIsAnimating=false; //是否处于动画状态标志位
    public Window2()
    { //构造函数,完成初始化工作
```

```
int i;
InitializeComponent();
//将 Xaml 文件声明的两个面板与列表容器配对加入字典
dicCanPairedUCList.Add(can1, lstCtr1);
dicCanPairedUCList.Add(can2, lstCtr2);
for(i=0; i< can1.Children.Count; i++)
{ //逐一将面板 1 中色相子实例入列,并保存标准位置数据
if (can1.Children[i].GetType() != typeof(UserControl1))
continue;
lstCtr.Add((UserControl1) can1.Children[i]);
((UserControl1) can1.Children[i]).ToTop = //ToTop 属性全程不变
(double)((UserControl1) can1.Children[i]).GetValue(Canvas.TopProperty);
lstXPos.Add((double)((UserControl1) can1.Children[i]).GetValue(Canvas.LeftProperty));
}
//逐一将面板 2 中色相子入列,代码从略
for(i=0; i<can2.Children.Count; i++) { ..... }
// DragFinished 在 Xaml 控件声明中关联至控件拖放行为的 DragEnd 事件
public void DragFinished(Object obj)
{
int i, j=0;
double dtemp, mindiff=10000;
//三行代码根据回调参数确定色相子控件、父面板及其配对列表容器
UserControl1 ctr = (UserControl1) obj;
Canvas can = (Canvas) VisualTreeHelper.GetParent(ctr);
List<UserControl1> lstCtrTemp = dicCanPairedUCList[can];
if(bIsNearIns) { //若是就近插入逻辑
for(i=0; i<lstCtrTemp.Count; i++) //查找控件释放时相距最近的标准位置
{ // * dtemp 用于保存受拖动控件与组内各控件的距离;距离最小的控件序号将保存在变量 j 中,代码从略 *
dtemp = Math.Abs(lstXPos.ElementAt(i) - (double)(ctr.GetValue(Canvas.LeftProperty)));
.....
}
//将被拖放的控件插入到新的位置
lstCtrTemp.Remove(ctr);
if(j == lstCtrTemp.Count) lstCtrTemp.Add(ctr);
else lstCtrTemp.Insert(j, ctr);
}
else { //空位侧优先插入逻辑直接采用 Lambda 表达式完成冒泡排序
lstCtrTemp.Sort((ctr1, ctr2) => {
return (double)ctr1.GetValue(Canvas.LeftProperty) <
(double)ctr2.GetValue(Canvas.LeftProperty) ? -1 : 1; });
}
```

```
//交换业务逻辑从略
.....
//创建故事板实例 sb 并行管理一组色相子的归位动画
Storyboard sb=new Storyboard();
for(i=0;i<lstCtrTemp.Count;i++)
{ //将标准 X 坐标赋给各色相子控件的 ToLeft
lstCtrTemp.ElementAt(i).ToLeft=lstXPos.ElementAt(i);
//横坐标归位动画
DoubleAnimation ctrAL=new DoubleAnimation();
ctrAL.To=lstCtrTemp.ElementAt(i).ToLeft;
ctrAL.Duration=TimeSpan.FromSeconds(0.1);
Storyboard.SetTarget(ctrAL,lstCtrTemp.ElementAt(i));
Storyboard.SetTargetProperty(ctrAL,new PropertyPath("(Canvas.Left)"));
sb.Children.Add(ctrAL);
//纵坐标归位动画与横坐标相类,代码从略
.....
}
//动画时长设置为 0.1 秒,以免拖延操作时间
sb.Duration=TimeSpan.FromSeconds(0.1);
//动画结束清理工作函数提交给故事板结束事件委托
sb.Completed+=storyboard_Completed;
sb.Begin();
storyboards.Add(can,sb); //将面板与动画配对加入字典
}
//动画结束时将被执行的清理函数
private void storyboard_Completed(object sender,EventArgs e)
{
//由结束的故事板时间线对象查找受控色相子的父面板及其配对列表容器,配对故事板实例
ClockGroup clockGroup=(ClockGroup)sender;
UserControl1 ctrTemp=(UserControl1)Storyboard.GetTarget(
(AnimationTimeline)clockGroup.Children[0].Timeline);
Canvas can=(Canvas)VisualTreeHelper.GetParent(ctrTemp);
List<UserControl1> lstctr=dicCanPairedUCList[can];
Storyboard storyboard=storyboards[can];
//停止、清除动画,并将面板、动画配对从字典中删除
storyboard.Stop();
storyboard.Children.Clear();
storyboards.Remove(can);
foreach(UserControl1 ctr in lstctr)
{ //逐一将同组各色相子的目标位置赋予自身的实际位置
属性
ctr.SetValue(Canvas.LeftProperty,ctr.ToLeft);
ctr.SetValue(Canvas.TopProperty,ctr.ToTop);
ctr.IsPressed=false; //清除控件拖放状态标志,去除层次效果
Canvas.SetZIndex(ctr,1); //恢复控件初始层次
}
}
// PreDrag 函数在 Xmal 声明中关联到控件拖放行为的 Pre-
```

```
Drag 事件
private bool PreDrag(object sender)
{
UserControl1 ctr=(UserControl1)sender;
//检查色相子控件的父面板是否处于动画状态
bool bTemp=storyboards.ContainsKey(((Canvas)ctr.Parent)) ? false:true;
if(bTemp){ //不在动画状态即将进入拖放操作
Canvas.SetZIndex(ctr,2); //提升控件叠放层次
ctr.IsPressed=true; //设置拖放标志,触发层次特效
}
return bTemp;
}
```

为色相子控件定义的 ToLeft、ToTop 依赖项属性作用在故事板中得到体现。如果不这样设计,就需要在主窗体中为全部色相子定义相应的依赖项属性数组,那样代码逻辑显然要混乱很多,并且当界面色相子控件数量发生变化时,也会带来代码维护上的不便。

本例中用到了 Dictionary<T,T>、List<T>两种泛型容器,带来的好处有两方面。首先,虽然 ArrayList 类可以实现与泛型类相似的功能,但泛型实例是强类型的,即在声明时必须给定容器内将要存储的对象类型,从而省去了 ArrayList 类应用时对未知数组元素的装箱/拆箱操作,提高程序运行效率;其次,泛型类直接或间接提供了对数组元素的查询、定位、插入、排序等丰富的成员函数或接口,在程序设计方面带来的便利是显而易见的。

3.4 与分辨率无关的自适应界面设计

对于 Win32 应用来说,控件随着界面的大小而自动缩放是一项繁琐的任务。但由于 WPF 强大的布局设计与图形化界面功能,这几乎不算一个问题。如图 3 中的可视化树所示,将面板包含在 Viewbox 中,并根据需要设置 Viewbox 的 Stretch、StretchDirection 属性,就可以实现控件随用户界面大小而自适应地动态调整自身大小,满足用户的视觉需求。如图 5 所示,对比标题栏大小可见两个不同大小窗口中,控件完全随着窗体的拉伸自适应缩放并调整布局绘制,不需要任何后台代码。

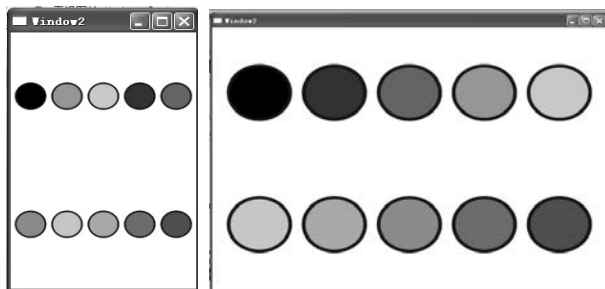


图5 仿真色相子界面自适应缩放及布局

这种与界面大小无关的控件自适应缩放与自动布局充分体现了 WPF 的强大, 其将编程人员从繁琐的界面布局逻辑中完全解放出来, 从而可以更加专注于核心业务代码设计, 对于程序开发效率和用户体验的提升具有划时代的意义。

4 结束语

根据 Farnsworth-Munsell 色相测试系统实物使用特点及原有软件的不足之处, 文中提出了基于 WPF 技术的色相子仿真交互界面设计方案。该方案充分利用到 .Net 的图形界面优势, 综合运用依赖项属性、触发器、故事板、泛型等 .Net 和 WPF 的优势特性, 具有界面外观可拓展性强、更接近实际物理特性的层次效果和仿真排序过渡动画、与分辨率无关的界面自适应缩放与布局调整等特点, 并对涉及的关键技术进行了详细介绍。在 Win32 应用逐渐落伍, 个性化、富媒体化跨平台应用日益崛起的大背景下, 文中介绍的 .Net 框架下的 WPF 相关技术, 对于图形化界面与人机交互类的应用开发都有一定的借鉴意义。文中示例在 Windows XP 系统 Visual Studio 2010 C# 开发环境下编译, Win XP、Win 8 操作系统测试通过。

参考文献:

- [1] Troelsen A. C# 与 .Net 4 高级程序设计[M]. 朱 晔, 肖 遼, 姚琪琳, 等, 译. 第 5 版. 北京: 人民邮电出版社, 2011.
- [2] MacDonald M. WPF 编程宝典—C#2010 版[M]. 王德才, 译. 北
- [6] Lee E K. A low voltage low output impedance CMOS bandgap voltage reference[C]//Proc of IEEE international symposium on circuits & systems. [s. l.]: IEEE, 2013: 1482–1483.
- [7] Camacho-Galeano E M, Galup-Montoro C, Schneider M C. A 2nW 1.1V self-biased current reference in CMOS technology[J]. IEEE Transactions on Circuits and Systems II, 2005, 52(2): 61–65.
- [8] Ahuja B K, Vu H, Laber C A, et al. A very high precision 500-nA CMOS floating-gate analog voltage reference[J]. IEEE Journal of Solid-state Circuits, 2005, 40(12): 2364–2372.
- [9] 王永顺, 井冰洁. 带有曲率补偿的高精度带隙基准电压源设计[J]. 半导体技术, 2014, 39(1): 14–18.
- [10] 吴贵能, 周 玮, 李儒章, 等. 通用二阶曲率补偿带隙基准电压源[J]. 微电子学, 2010, 40(2): 204–208.
- [11] 来新泉, 郝 琦, 袁 冰, 等. 一种二阶曲率补偿的高精度带隙基准电压源[J]. 西安电子科技大学学报: 自然科学

京: 清华大学出版社, 2011.

- [3] Petzold C. Windows Presentation Foundation 程序设计指南[M]. 蔡学镛, 译. 北京: 电子工业出版社, 2008.
- [4] 王 鹏, 崔 静. 新一代界面技术 WPF 的架构及应用[J]. 成都纺织高等专科学校学报, 2011, 28(1): 18–20.
- [5] 张京明, 曹国忠, 张承业, 等. 创新方法与交互设计结合的探讨[J]. 价值工程, 2012, 31(10): 10–11.
- [6] 李 颖. 基于 WPF 的课堂教学管理系统研究与设计[J]. 中国教育技术装备, 2011(24): 85–87.
- [7] 袁云云, 周之昊. 基于 WPF 的医疗辅助软件设计与开发[J]. 数字技术与应用, 2012(2): 132–132.
- [8] 李成刚, 冯 静, 凌 玲. 基于 WPF 的交互式绘图系统的开发[J]. 微型机与应用, 2011, 30(6): 50–52.
- [9] Macdonald M. Pro WPF in C# 2008; Windows Presentation Foundation with .NET 3.5[M]. New York: Apress, 2008.
- [10] Farnsworth D. The Farnsworth-Munsell 100 hue and Dichotomous tests for color vision[J]. Journal of the Optical Society of America B-optical Physics, 1943, 33: 568–578.
- [11] Hidajat R R, Hidayat J R, McLay J L, et al. A fast system for reporting the Farnsworth - Munsell 100-hue colour vision test[J]. Documenta Ophthalmologica, 2004, 109(2): 109–114.
- [12] Ghose S, Parmar T, Dada T, et al. A new computer-based Farnsworth-Munsell 100-hue test for evaluation of color vision[J]. International Ophthalmology, 2014, 34(4): 747–751.
- [13] Melamud A, Simpson E, Traboulsi E I. Introducing a new computer-based test for the clinical evaluation of color discrimination[J]. American Journal of Ophthalmology, 2006, 142(6): 953–960.
- [14] 张洪定, 孟冬梅. 基于 Expression Blend 4 中文版 WPF 和 Silverlight 项目设计基础[M]. 北京: 清华大学出版社, 2011.


版, 2010, 37(5): 911–915.

- [12] 张宗航, 赵毅强, 耿俊峰. 一种二阶曲率补偿带隙基准电压源[J]. 微电子学与计算机, 2012, 29(5): 15–19.
- [13] 杨 鹏, 吴志明, 吕 坚, 等. 一种二阶补偿的低压 CMOS 带隙基准电压源[J]. 微电子学, 2007, 37(6): 891–894.
- [14] Du F M A M. A 5.8ppm/°C bandgap reference with a preregulator[C]//Proc of ICEE. [s. l.]: [s. n.], 2011.
- [15] Dey A, Bhattacharyya T K. A CMOS bandgap reference with high PSRR and improved temperature stability for system-on-chip applications [C]//Proc of EDSSC. [s. l.]: [s. n.], 2011: 17–21.
- [16] Khan Q A, Wadhwa S K, Misri K. A low voltage switched-capacitor current reference circuit with low dependence on process, voltage and temperature [C]//Proceedings of the 16th international conference on VLSI design. [s. l.]: [s. n.], 2003: 504–506.

WPF技术在蒙塞尔色相仿真测试中的应用

作者：[李晓京](#)，[马进](#)，[胡文东](#)，[张利利](#)，[惠铎铎](#)，[LI Xiao-jing](#)，[MA Jin](#)，[HU Wen-dong](#)，[ZHANG Li-li](#)，[HUI Duo-duo](#)

作者单位：[第四军医大学 航空航天医学教育部重点实验室, 陕西 西安, 710032](#)

刊名：[计算机技术与发展](#)

英文刊名：

年，卷(期)：2016 (2)

引用本文格式：[李晓京](#).[马进](#).[胡文东](#).[张利利](#).[惠铎铎](#).[LI Xiao-jing](#).[MA Jin](#).[HU Wen-dong](#).[ZHANG Li-li](#).[HUI Duo-duo](#) [WPF技术在蒙塞尔色相仿真测试中的应用](#)[期刊论文]-[计算机技术与发展](#) 2016 (2)