Windows Phone Mango编程实践

***Windows Phone Mango Programming Practice***

第一篇 基础篇

# 多点触控

《周易》曰：“泽中有火，革。君子以治历明时。”意思是汪洋大泽中有烈火，象征着急需变革。君子当效法于此，改革立法来明确季节的变化，按规律行事。在科技革故鼎新面前，君子应明辨发展方向，及时掌握前沿技术，多点触控技术就是一场触控技术的革命，对智能手机的应用形态和产品形态产生了重大的影响，作为Windows Phone的开发者应在应用程序中最大限度的发挥多点触控的功能。

## 概述

《周易》曰：“泽中有火，革。君子以治历明时。”意思是汪洋大泽中有烈火，象征着急需变革。君子当效法于此，改革立法来明确季节的变化，按规律行事。在科技革故鼎新面前，君子应明辨发展方向，及时掌握前沿技术，多点触控技术就是一场触控技术的革命，对智能手机的应用形态和产品形态产生了重大的影响，作为Windows Phone的开发者应在应用程序中最大限度的发挥多点触控的功能。

多点触控技术是一场触控技术方面核心是FTIR（Frustrated Total Internal Reflection），即受抑内全反射技术。由LED（发光二极管）发出的光束从触摸屏截面照向屏幕的表面后，将产生反射。如果屏幕表层是空气，当入射光的角度满足一定条件时，光就会在屏幕表面完全反射。但是如果有个折射率比较高的物质(例如手指)压住丙烯酸材料面板，屏幕表面全反射的条件就会被打破，部分光束透过表面，投射到手指表面。凹凸不平的手指表面导致光束产生散射（漫反射），散射光透过触摸屏后到达光电传感器，光电传感器将光信号转变为电信号，操作系统由此获得相应的触摸信息。

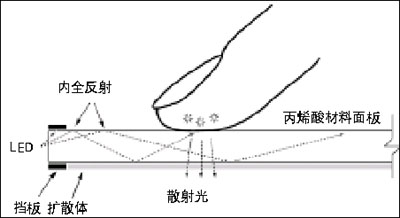


图3-1 受抑内全反射技术

触控的输入方式是Windows Phone的核心功能，手指触控的反馈提供智能手机使用者更多难忘的探索期望和乐趣无穷的交互式体验。例如，拖拽照片或者滑动手指轻轻翻页，而多点触控识别和处理则提供更多难以想象的功能。

触控键盘声音：用户在触控Windows Phone的虚拟键盘时，会获得不一样的体验。Windows Phone系统循环发出8种不同的声音，就像从远处传来的脚步声，虽然相似却不同，以此自然的效果来减少用户重复按键的“焦虑”。

Silverlight和XNA对于触控的处理却不尽然相同，Silverlight的触控识别通过捕获事件的方式实现的，在XNA的触控识别是通过静态类的循环周期的轮询实现的。XNA Update方法的主要目的就是检查触控的状态，并将其响应效果通过Draw方法反应在屏幕上。

令人兴奋的是，在Windows Phone Mango支持Silverlight和XNA的整合，在Silverlight中可以使用XNA更为强大和复杂的触控识别和处理。

## Windows Phone支持的触控指令

| **手势名称** | **图例** | **说明** |
| --- | --- | --- |
| Tap |  | 选择对象或停止从屏幕上移动任何内容。 |
| Double tap |  | 连续触碰。连续触碰同一个位置两次。 |
| Pan |  | 拖拽屏幕上的一个对象到一个不同的位置。 |
| Flick |  | 向任何方向移动整个画布。 |
| Touch and hold |  | 触碰后持续达一段时间，显示上下文菜单或选项页的内容。 |
| Pinch |  | 缩小或减少对象（取决于应用）。 |
| Stretch |  | 放大或扩大对象（取决于应用）。 |

表3-1 触控指令

## 动手实践——Silverlight的多点触控

告别简单的手势识别，通过对复杂的手势的反馈带给操作者的感受无异于驾驭F1驰骋。

### 动手实践

1. 新建一个Visual C#的Windows Phone工程，工程名称为ManipulationProject。

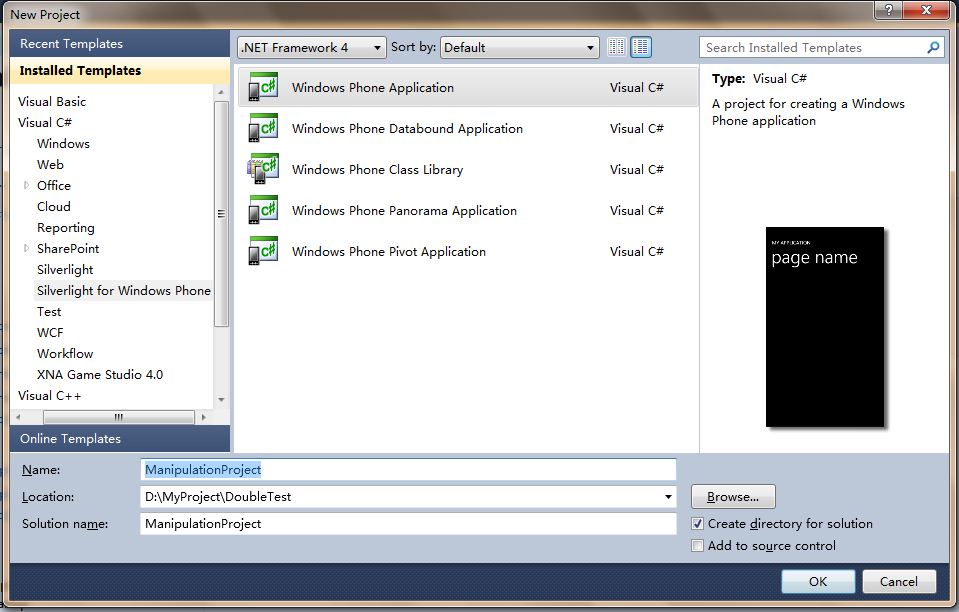


图3-2 新建工程

1. 将下面的XAML代码加入到MainPage.xaml中。

这段代码的作用是在画布上创建了一个蓝色的的矩形。应用程序订阅了ManipulationDelta事件，对于该事件的响应代码中包含的移动矩形的控制逻辑。

Silverlight Project: ManipulationProject File: MainPage.xaml

<Canvas>

<Rectangle

Name="rectangle"

Width="200" Height="200"

Fill="Blue" Stroke="Blue" StrokeThickness="1" />

</Canvas>

1. 在MainPage类中, 添加如下的变量。

Silverlight Project: ManipulationProject File: MainPage.xaml.cs

private TransformGroup transformGroup;

private TranslateTransform translation;

private ScaleTransform scale;

1. 在MainPage.xaml中添加[ManipulationDelta](http://msdn.microsoft.com/zh-cn/library/system.windows.uielement.manipulationdelta(v=VS.95).aspx?appId=Dev10IDEF1&l=ZH-CN&k=k(SYSTEM.WINDOWS.SHAPES.RECTANGLE);k(VS.XAMLEDITOR);k(TargetFrameworkMoniker-)事件处理程序Canvas\_ManipulationDelta。

如图在MainPage.xaml中点击右键，选择”Properties”。

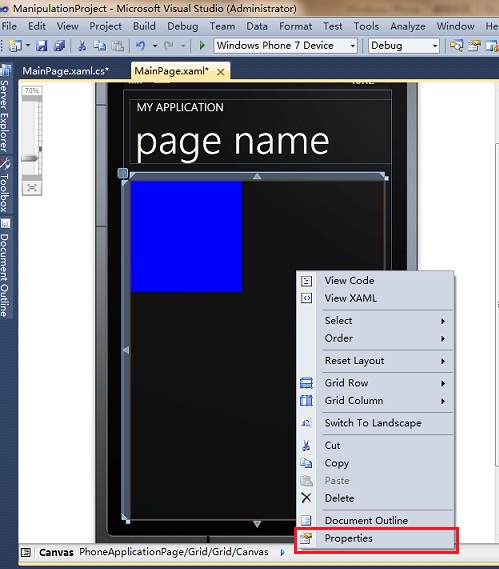


图3-3 属性

在”Event”中设置ManipulationDelta的响应事件Canvas\_ManipulationDelta。

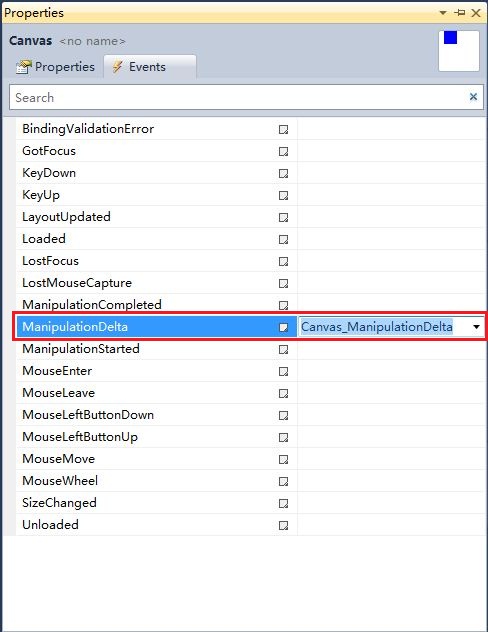


图3-4 修改属性

设置完毕后，我们会在MainPage类中看见下面的代码被自动添加：

Silverlight Project: ManipulationProject File: MainPage.xaml.cs

private void Canvas\_ManipulationDelta(object sender, ManipulationDeltaEventArgs e)

{

}

1. 添加下面的代码到MainPage的构造函数中。

this.transformGroup = new TransformGroup();

this.translation = new TranslateTransform();

this.scale = new ScaleTransform();

this.transformGroup.Children.Add(this.scale);

this.transformGroup.Children.Add(this.translation);

this.rectangle.RenderTransform = this.transformGroup;

在MainPage类，ManipulationDelta事件处理程序Canvas\_ManipulationDelta中添加以下代码。

在手指操作期间ManipulationDelta事件发生时触摸输入更改位置可能发生多次。例如，如果用户手指拖动屏幕，ManipulationDelta事件发生多次作为手指移动。

Silverlight Project: ManipulationProject File: MainPage.xaml.cs

private void Canvas\_ManipulationDelta(object sender, ManipulationDeltaEventArgs e)

{

// Scale the rectangle.

if ((e.DeltaManipulation.Scale.X == 0) || (e.DeltaManipulation.Scale.Y == 0))

{

// Increase ScaleX and ScaleY by 5%.

this.scale.ScaleX \*= 1.05;

this.scale.ScaleY \*= 1.05;

}

else

{

this.scale.ScaleX \*= e.DeltaManipulation.Scale.X;

this.scale.ScaleY \*= e.DeltaManipulation.Scale.Y;

}

// Move the rectangle.

this.translation.X += e.DeltaManipulation.Translation.X;

this.translation.Y += e.DeltaManipulation.Translation.Y;

}

1. 编译并运行程序。

屏幕上蓝色的矩形等着您的触摸。

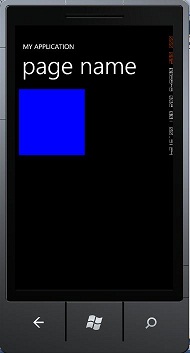


图3-5 运行结果

### 测试应用程序

试一试复杂的手势变化，尝试Windows Phone的互动体验。

* 移动矩形，你的手指按住该矩形，并在屏幕上移动手指。
* 若要调整矩形的形状，把两个手指放在矩形上缩小和延伸（Pinch and Stretch），如下图。



图3-6 手势识别

## 耀眼的火花——XNA多点触控游戏

手指划过屏幕产生的火花，就像夜空中燃放的烟火，瞬间产生，留下光芒后渐渐消逝。



图3-7 应用程序运行效果

### 创建应用程序

新建一个XNA Game Studio 4.0的Windows Phone Game(4.0)的工程，工程名称为InputToyWP7。

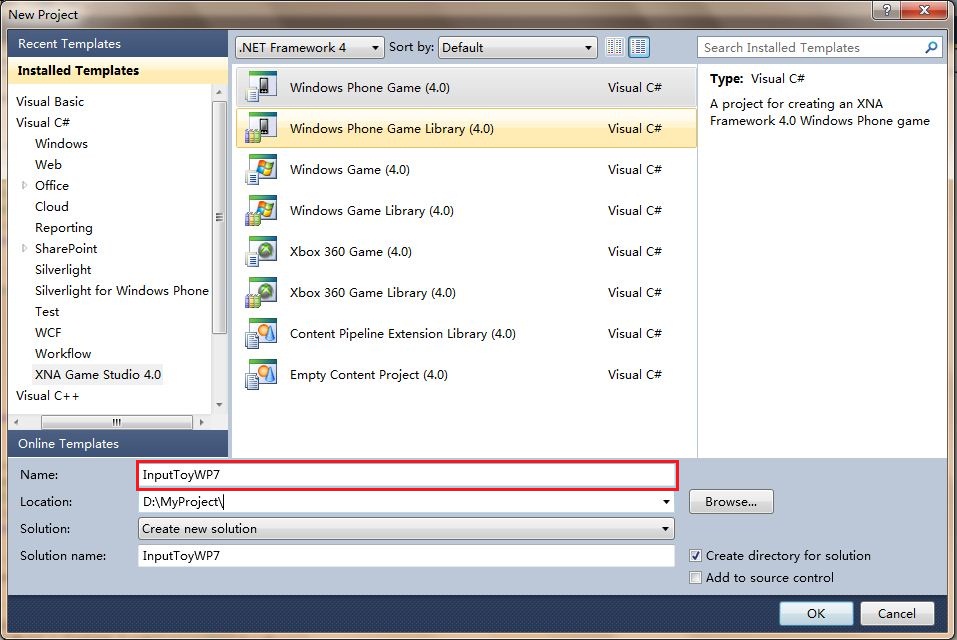


图3-8 新建工程

### 启用手势操作支持

以 XNA 为基础的游戏程序必须设定 TouchPanel 类别的 EnabledGestures 属性，才能够启用手势操作功能，以支持用户以手势操作游戏程序。

程序设计师可以在 Game1 类别的 Initialize 方法执行设定 TouchPanel 类别的 EnabledGestures 属性的动作，以启用手势操作支持，做法如下：

XNA Project: ManipulationProject File: Game1.cs

// initialization logic

TouchPanel.EnabledGestures =

GestureType.Hold |

GestureType.Tap |

GestureType.DoubleTap |

GestureType.FreeDrag |

GestureType.HorizontalDrag |

GestureType.Flick |

GestureType.Pinch;

**注意：**

以 XNA 为基础的游戏程序必须启用触控功能才能够让游戏的使用者进行触控操作，如果已启用 Pinch 操作功能，则当用户利用两个手指头同时触碰触控屏幕并进行移动时，就会产生 Pinch 操作，而不是两个不同的拖曳操作，如果未启用 Pinch 操作功能，则所产生的就不是 Pinch 操作，而是依据两个触碰位置的平均为准的单一拖曳操作。

下面的代码演示获取手机支持的触控个数。

XNA Project: ManipulationProject File: Game1.cs

TouchPanelCapabilities tc = TouchPanel.GetCapabilities();

if (tc.IsConnected)

{

return tc.MaximumTouchCount;

}

**注意：**

Windows Phone的XNA Game Studio 4.0支持的触控数为4个。

### 处理使用者的手势操作

启用了手势操作功能之后，以 XNA 为基础的应用程序可以在 Game1 类别的 Update 方法中呼叫 TouchPanel 类别的 ReadGesture 方法取得用户的手势操作信息。请注意读取使用者的手势操作的做法和呼叫 TouchPanel 类别的 GetState 方法读取触控面板的状态的做法不同，因为用户对游戏程序的触控操作会产生多个手势信息，来不及被游戏处理的手势信息会被存放到队列中等待处理，让游戏程序利用循环取出并加以处理。

以下的 Update 方法便会利用 while 循环，搭配 TouchPanel 类别的 IsGestureAvailable 属性判断是否还有用户触控操作产生的手势信息尚未被处理，如果尚有使用者触控操作产生的手势信息尚未被处理，则呼叫 TouchPanel 类别的 ReadGesture 方法读取手势信息，并加以处理，修改Update方法的代码如下。

XNA Project: ManipulationProject File: Game1.cs

protected override void Update(GameTime gameTime)

{

// 允许游戏退出

if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)

{

this.Exit();

}

while (TouchPanel.IsGestureAvailable)//判断是否有手势信息未处理

{

GestureSample gesture = TouchPanel.ReadGesture();

switch (gesture.GestureType)

{

case GestureType.Tap:

case GestureType.DoubleTap:

case GestureType.Hold:

case GestureType.FreeDrag:

case GestureType.Flick:

case GestureType.Pinch:

// 处理手势信息

……

break;

}

}

base.Update(gameTime);

}

因为 GestureType.FreeDrag 自由拖曳操作已经包括 GestureType.VerticalDrag 垂直拖曳操作和 GestureType.HorizontalDrag 水平拖曳操作，所以游戏程序在判断用户的触控操作的动作时，不需要既判断动作是否为 GestureType.FreeDrag，又判断动作是否为 GestureType.VerticalDrag 或 GestureType.HorizontalDrag，两者择一处理即可。

**[特别注意]**

使用 TouchPanel 类别进行触控控制的游戏程序可以呼叫 TouchPanel 类别的 GetState 方法取得使用者对触控面板的触控状态，或是呼叫 TouchPanel 类别的 ReadGesture 方法取得使用者的手势操作状态，不要两者混用，否则将会无法得到正确的结果。例如先利用 TouchPanel 类别的 ReadGesture 方法取得使用者的手势操作状态，再利用 TouchPanel 类别的 GetState 方法取得用户触碰屏幕的位置，当做手势操作触碰屏幕的位置来使用就是错误的做法。

请注意呼叫 TouchPanel 类别的 ReadGesture 方法读取手势信息时，读取到的手势信息会以 GestureSample 结构的型式传回给呼叫者。表2 所示为 GestureSample 结构常用的属性：

|  |  |
| --- | --- |
| 属性名称 | 说明 |
| Delta | 存放与第一个碰触点的偏差量。 |
| Delta2 | 存放与第二个碰触点的偏差量。 |
| GestureType | 存放触控操作的种类。 |
| Position | 存放第一个碰触点的位置。 |
| Position2 | 存放第二个碰触点的位置。 |
| Timestamp | 存放触控操作发生的时间。 |

表3-2 GestureSample 结构常用的属性

因为呼叫 TouchPanel 类别的 ReadGesture 方法传回的 GestureSample 结构只能存放第一个和第二触碰点的位置：Position 和 Postition2，以及存放两个触碰点的偏差量，所以只能支持到最多两个手指头的操作。

请注意 GestureSample 结构的属性的型态为 TimeSpan 结构而不是 GameTime 类别，用来代表手势操作与手势操作之间的时间间隔。

各种手势操作需要用到的 GestureSample 结构的属性可以参考下表的详细说明：

|  |  |
| --- | --- |
| **触控操作** | **说明** |
| **Tap** | Position 属性 |
| **DoubleTap** | Position 属性 |
| **Hold** | Position 属性 |
| **VerticalDrag** | Position 属性和 Delta 属性 |
| **HorizontalDrag** | Position 属性和 Delta 属性 |
| **FreeDrag** | Position 属性和 Delta 属性 |
| **DragComplete** | 无 |
| **Flick** | Delta 属性 |
| **Pinch** | Position、Position2、Delta、和 Delta 2 四个属性 |
| **PinchComplete** | 无 |

表3-3 各种手势操作需要用到的 GestureSample 结构的属性

### 处理手势操作的要诀

在处理使用者的触控操作方面，Flick 触控操作可以利用 GestureSample 结构的 Delta 属性的内容值当做用户轻拂的快慢速度，以控制卷动游戏内容的速度。VerticalDrag 和 HorizontalDrag 触控操作可以利用 GestureSample 结构的 Delta 属性的内容值判断垂直和水平移动的距离，而且 VerticalDrag 触控操作的 Delta 属性的 X 成员的内容值必为 0，而 HorizontalDrag 触控操作的 Delta 属性的 Y 成员的内容值必为 0，不需要另外透过程序代码进行设定。Pinch 触控操作是两个手指头并用的触控操作，常常用来执行旋转对象、放大缩小对象、或是旋转相机镜头的动作。游戏程序可以利用 GestureSample 结构的 Position 属性和 Delta 属性，取得第一个手指头的触碰点和偏差量，利用 Position2 属性和 Delta2 属性取得第二个手指头的触碰点和偏差量，再据以执行变更游戏程序显示的内容的动作。

### 读取多点触控的数据

如果需要取得用户同时触控的状态，则可以呼叫 TouchPanel 类别的 GetState 方法，取得所有触碰点的集合，再利用循环取出集合中的所有触碰点并加以处理，做法如下：

TouchCollection touchCollection = TouchPanel.GetState();//取得触摸屏幕状态

foreach (TouchLocation tl in touchCollection)

{

//判断状态

……

}

### 设计支持手势操作的 XNA 游戏

了解 XNA Framework 支持触控操作的基本功能之后，接下来我们就要设计一个能够允许用户利用触控屏幕操作的简单游戏。

1. 将InputToyWP7示例的Menu.cs和Instructions.cs添加到工程中；
2. 将InputToyWP7示例的GameThumbnail.png图片替换新建程序中的图片；
3. 在InputToycontet中添加程序所用图片，在Solution Explorer 上的InputToycontet(content) 上点击右键，选择Add…，具体操作如图。

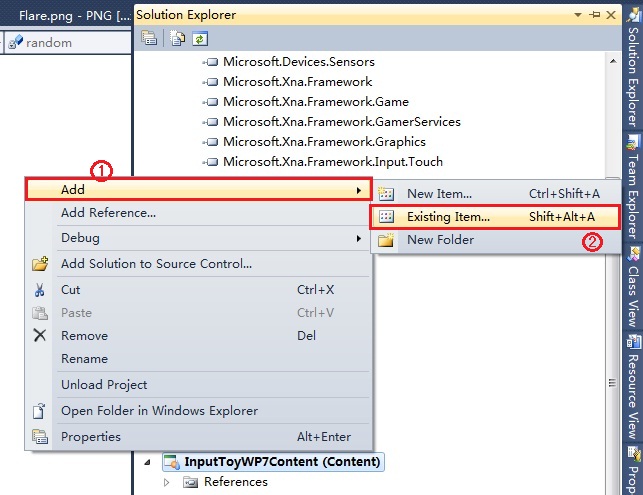


图3-9 添加资源

1. 在Game1的类中修改游戏控制逻辑。

XNA Project: ManipulationProject File: Game1.cs

public class Game1 : Microsoft.Xna.Framework.Game

{

static Random random = new Random();

class Sparkle

{

public Vector2 position;

public Vector2 speed;

public float rotation; // in radians

public Color color;

public long birthtime;

public Sparkle(float x, float y, long time)

{

position = new Vector2(x, y);

rotation = (float)(random.NextDouble() \* 2.0 \* Math.PI);

color = Color.White;

birthtime = time;

}

}

const int SPARKLELIFE = 4000; // ticks

const int MAXSPARKLES = 100; // maximum number of sparkles, for

// performance tuning.

const float ACCELFACTOR = 0.01f;

const float FADEFACTOR = 255.0f / SPARKLELIFE;

Vector2 screenDimensions = new Vector2(272, 480);

GraphicsDeviceManager graphics;

SpriteBatch spriteBatch;

Texture2D flareImage;

Vector2 flareOffset;

List<Sparkle> sparkles;

Instructions instructions;

Menu menu;

bool accelActive = false;

Accelerometer accelSensor;

Vector3 accelReading = new Vector3();

public Game1()

{

graphics = new GraphicsDeviceManager(this);

sparkles = new List<Sparkle>();

instructions = new Instructions();

menu = new Menu();

Content.RootDirectory = "Content";

// 默更新频率30 fps

TargetElapsedTime = TimeSpan.FromSeconds(1 / 30.0);

// 使硬件内置的分辨率与原始的(Zune HD)一致

graphics.PreferredBackBufferWidth = (int)screenDimensions.X;

graphics.PreferredBackBufferHeight = (int)screenDimensions.Y;

// 全屏

graphics.IsFullScreen = true;

accelSensor = new Accelerometer();

// 为重力加速传感器添加事件处理函数

accelSensor.ReadingChanged +=

new EventHandler<AccelerometerReadingEventArgs>(AccelerometerReadingChanged);

// 启动重力加速传感

try

{

accelSensor.Start();

accelActive = true;

}

catch (AccelerometerFailedException e)

{

// 重力加速传感启动失败

accelActive = false;

}

catch (UnauthorizedAccessException e)

{

// 抛出模拟器不支持重力加速传感的异常

accelActive = false;

}

}

/// <summary>

/// Allows the game to perform any initialization it needs to before

/// starting to run. This is where it can query for any required

/// services and load any non-graphic related content. Calling

/// base.Initialize will enumerate through any components and

/// initialize them as well.

/// </summary>

protected override void Initialize()

{

// TODO: Add your initialization logic here

TouchPanel.EnabledGestures =

GestureType.Hold |

GestureType.Tap |

GestureType.DoubleTap |

GestureType.FreeDrag |

GestureType.HorizontalDrag |

GestureType.Flick |

GestureType.Pinch;

base.Initialize();

}

/// <summary>

/// LoadContent will be called once per game and is the place to load

/// all of your content.

/// </summary>

protected override void LoadContent()

{

// Create a new SpriteBatch, which can be used to draw textures.

spriteBatch = new SpriteBatch(GraphicsDevice);

flareImage = this.Content.Load<Texture2D>("Flare");

flareOffset = new Vector2(

flareImage.Width \* 0.5f, flareImage.Height \* 0.5f);

menu.loadContent(this.Content, this.GraphicsDevice);

instructions.loadContent(this.Content, this.GraphicsDevice);

instructions.show();

}

/// <summary>

/// UnloadContent will be called once per game and is the place to

/// unload all content.

/// </summary>

// 卸载全部内容

protected override void UnloadContent()

{

// Unload any non ContentManager content here

// 如果传感器启动，则将其停止

if (accelActive)

{

try

{

accelSensor.Stop();

}

catch (AccelerometerFailedException e)

{

// 捕获传感器无法停止的异常

}

}

}

/// <summary>

/// Allows the game to run logic such as updating the world,

/// checking for collisions, gathering input, and playing audio.

/// </summary>

/// <param name="gameTime">Provides a snapshot of timing values.</param>

protected override void Update(GameTime gameTime)

{

// 允许游戏退出

if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)

{

this.Exit();

}

long ttms = (long)gameTime.TotalGameTime.TotalMilliseconds;

// ElapsedGameTime过去的时间

float etms = gameTime.ElapsedGameTime.Milliseconds;

// 更新instructions状态

if (instructions.isVisible())

{

instructions.update(etms);

}

while (TouchPanel.IsGestureAvailable)

{

GestureSample gesture = TouchPanel.ReadGesture();

switch (gesture.GestureType)

{

case GestureType.Tap:

case GestureType.DoubleTap:

case GestureType.Hold:

case GestureType.FreeDrag:

case GestureType.Flick:

case GestureType.Pinch:

TouchCollection touchCollection = TouchPanel.GetState();

foreach (TouchLocation tl in touchCollection)

{

if ((tl.State == TouchLocationState.Pressed)

|| (tl.State == TouchLocationState.Moved))

{

if (menu.handleInput(tl, instructions))

{

continue;

}

// 在触控的位置添加火花

sparkles.Add(new Sparkle(tl.Position.X,

tl.Position.Y, ttms));

// 火花数量大于最大值的处理：删除最早添加的火花

if (sparkles.Count > MAXSPARKLES)

{

sparkles.RemoveAt(0);

}

}

}

break;

}

}

// 更新火花的状态

for (int i = sparkles.Count - 1; i >= 0; i--)

{

Sparkle s = sparkles[i];

if (!menu.isPaused() && ((ttms - s.birthtime) > SPARKLELIFE))

{

sparkles.RemoveAt(i);

continue;

}

else

{

if (menu.isPaused())

{

// 当暂停时，重置发生时间，以便火花不会立即消失。

s.birthtime = ttms - SPARKLELIFE

+ (long)(s.color.A / FADEFACTOR);

}

else

{

// 淡入淡出火花的闪耀效果

s.color.A = (byte)(255.0f - (ttms - s.birthtime)

\* FADEFACTOR);

}

// 暂停时火花依然可以移动

if (accelActive)

{

// 火花的加速度取决于加速感应动作

s.speed.X += accelReading.X \* ACCELFACTOR;

s.speed.Y += -accelReading.Y \* ACCELFACTOR;

// 以速度移动火花的位置

s.position.X += s.speed.X \* etms;

s.position.Y += s.speed.Y \* etms;

s.rotation += s.speed.Length() \* ACCELFACTOR \* etms;

}

}

}

base.Update(gameTime);

}

/// <summary>

/// This is called when the game should draw itself.

/// </summary>

/// <param name="gameTime">Provides a snapshot of timing values.</param>

protected override void Draw(GameTime gameTime)

{

GraphicsDevice.Clear(Color.Black);

spriteBatch.Begin();

// 绘制背景

if (instructions.isVisible())

{

instructions.draw(spriteBatch);

}

// 绘制火花

foreach (Sparkle s in sparkles)

{

// when drawing with a specified origin, the origin affects

// both the rotation and position parameters.

spriteBatch.Draw(

flareImage, s.position, null, s.color, s.rotation,

flareOffset, 1.0f, SpriteEffects.None, 0);

}

// 绘制背景

menu.draw(spriteBatch);

spriteBatch.End();

base.Draw(gameTime);

}

public void AccelerometerReadingChanged(object sender, AccelerometerReadingEventArgs e)

{

accelReading.X = (float)e.X;

accelReading.Y = (float)e.Y;

accelReading.Z = (float)e.Z;

}

public int getTouchPoints()

{

TouchPanelCapabilities tc = TouchPanel.GetCapabilities();

if(tc.IsConnected)

{

return tc.MaximumTouchCount;

}

else

{

return -1;

}

}

}

### 游戏程序部署

程序设计师可以利用 Visual Studio 2010 Express for Windows Phone 直接将所开发的程序部署到 Windows Phone智能手机，不过程序设计师所使用的计算机仍然必须事先安装好 Zune 软件。程序设计师可以利用Visual Studio工具栏提供的下拉选项 [] 选择Windows Phone Device，再按下 CTRL+F5 组合键将开发好的程序部署到 Windows Phone智能手机。同样地，Windows Phone智能手机不可以处于屏幕锁定的状态，否则将无法部署成功。

请执行部署到 Windows Phone智能手机的游戏程序，并利用 XNA Framework 支持的各种手势操作技巧点选、自由移动、轻拂、或放大缩小游戏程序显示的饼图案，体验利用手势操作程序的高度方便性。

手指划过屏幕产生的火花，就像夜空中燃放的烟火，瞬间产生，留下光芒后渐渐消逝，仿佛人生匆匆走过，留下精彩，留下脚下的路，挥挥手。