# LỜI CẢM ƠN

# NHẬN XÉT GIẢNG VIÊN HƯỚNG DẪN

Thành phố Hồ Chí Minh, ngày tháng năm 2016

Giảng viên hướng dẫn

Mục lục

[1. Tổng quan về WPF 1](#_Toc280620110)

[1.1. Nền tảng thống nhất để xây dựng giao diện người dùng 1](#_Toc280620111)

[1.2. Khả năng làm việc giữa người thiết kế giao diện và lập trình viên. 2](#_Toc280620112)

[1.3. Các đặc điểm tạo ra sự khác biệt của WPF 3](#_Toc280620113)

[2. Đồ họa 2D trong WPF 4](#_Toc280620114)

[2.1. Các đối tượng hình học cơ bản-Shape 4](#_Toc280620115)

[2.2. Biến đổi hình học – Transfrom 4](#_Toc280620116)

[3. Sử dụng chổi tô – Brush 6](#_Toc280620117)

[3.1. Tô màu đồng nhất – Solid Color 7](#_Toc280620118)

[3.2. Tô màu theo định hướng tuyến tính – Linear Gradient Color 8](#_Toc280620119)

[3.3. Đổ màu theo bán kinh hình tròn – Radial Gradient 10](#_Toc280620120)

[3.4. Tô màu bằng Bitmap 11](#_Toc280620121)

[3.5. Thiết lập độ mờ của chổi tô – Opacity 13](#_Toc280620122)

[4. Animation 14](#_Toc280620123)

[4.1. Basic Animation 15](#_Toc280620124)

[4.2. Các lớp Animation 16](#_Toc280620125)

[4.3. Animation trong code 17](#_Toc280620126)

[4.4. Animation kép – Simultaneuos Animations 20](#_Toc280620127)

[4.5. The Timeline Class 20](#_Toc280620128)

[4.6. Animation và Storyboards 24](#_Toc280620129)

[4.6.1. Storyboard 24](#_Toc280620130)

[4.6.2. Event trigger 25](#_Toc280620131)

[4.6.3. Gắn Trigger với Style 25](#_Toc280620132)

[4.6.4. Gắn Triggers với Template 26](#_Toc280620133)

[4.6.5. Điều khiển các kĩ thuật phát lại – Playback 27](#_Toc280620134)

[4.7. Kiểu Animation Revisited 28](#_Toc280620135)

[5. Kết luận 28](#_Toc280620136)

[5.1 Kết quả đạt được 28](#_Toc280620137)

[5.2 Hạn chế của luận văn 29](#_Toc280620138)

[6. Kết quả demo minh họa 30](#_Toc280620139)

# 

# 1. Tổng quan về WPF

Trong các ứng dụng hiện đại, giao diện người dùng trực quan chiếm vị trí hết sức quan trọng. Việc trình diễn đúng thông tin, theo đúng cách và vào đúng thời điểm có thể đem lại những giá trị kinh tế xã hội đáng kể. Với những ứng dụng thương mại, chẳng hạn một ứng dụng bán hàng trực tuyến, việc cung cấp một giao diện người dùng mạnh có thể tạo nên sự khác biệt giữa một công ty với các đối thủ cạnh tranh, góp phần làm tăng doanh số và giá trị thương hiệu của hãng này so với hãng khác. Để có được một giao diện người dùng như vậy, việc tích hợp đồ họa, media, văn bản và các thành phần trực quan khác như một thể thống nhất đóng đóng vai trò mấu chốt.

Với công nghệ 2006 việc tạo ra một giao diện đẹp như đã nói ở trên trên nền tảng windown là một việc hết sức khó khăn bởi những vấn đề sau:

* Có rất nhiều công nghệ khác nhau được sử dụng để làm việc với hình ảnh âm thanh và video. Tìm được những lập trình viên có khả năng sử dụng tốt nhiều công nghệ như vậy không dễ và chi phí cao cho cả quá trình phát triển cũng như bảo trì ứng dụng.
* Thiết kế một giao diện biểu diễn có hiệu quả tất cả những tính năng như vậy cũng là một thách thức. Nó đòi hỏi phải có những người thiết kế giao diện chuyên nghiệp, bởi lập trình viên phần mềm đơn thuần sẽ không có đủ các kỹ năng cần thiết. Điều này lại dẫn tới những khó khăn phát sinh khi người thiết kế và người lập trình làm việc chung.
* Việc cung cấp một giao diện đầy đủ tính năng, hoạt động được như một ứng dụng Windows riêng biệt trên máy desktop, đồng thời có thể được truy nhập thông qua trình duyệt có thể đòi hỏi phải xây dựng hai phiên bản độc lập sử dụng hai công nghệ khác nhau. Ứng dụng Windows trên desktop sử dụng Windows Forms và các công nghệ thuần Windows khác, trong khi ứng dụng trên trình duyệt lại sử dụng HTML và JavaScript. Do đó, cần phải có hai nhóm phát triển với hai phần kỹ năng khác nhau.

## 1.1. Nền tảng thống nhất để xây dựng giao diện người dùng

Để tạo form, các control và các tính năng kinh điển khác của một giao diện đồ họa Windows, thông thường lập trình viên sẽ chọn Windows Forms, một phần của .NET Framework. Nếu cần hiển thị văn bản, Windows Forms có một số tính năng hỗ trợ văn bản trực tiếp hoặc có thể sử dụng Adobe’s PDF để hiển thị văn bản có khuôn dạng cố định. Đối với hình ảnh và đồ họa 2 chiều, lập trình viên sẽ dùng GDI+, một mô hình lập trình riêng biệt có thể truy nhập qua Windows Forms. Để hiển thị video hay phát âm thanh, lập trình viên lại phải sử dụng Windows Media Player, và với đồ họa 3 chiều, ta lại phải dùng Direct3D, một thành phần chuẩn khác của Windows. Tóm lại, quá trình phát triển giao diện người dùng theo yêu cầu trở nên phức tạp, đòi hỏi lập trình viên quá nhiều kỹ năng công nghệ.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Windows  Forms | PDF | Windows  Forms/  GDI+ | Windows  Media  Player | Direct3D | WPF |
| Giao diện đồ họa (form và các control) | x |  |  |  |  | x |
| On-screen văn bản | x |  |  |  |  | x |
| Fixed-format văn bản |  | x |  |  |  | x |
| Hình ảnh |  |  | x |  |  | x |
| Video và âm thanh |  |  |  | x |  | x |
| Đồ họa 2 chiều |  |  | x |  |  | x |
| Đồ họa 3 chiều |  |  |  |  |  | x |

WPF là giải pháp hợp nhất nhằm giải quyết tất cả những vấn đề công nghệ nêu trên, hay nói cách khác, WPF cung cấp nhiều tính năng lập trình giao diện trong cùng một công nghệ đơn nhất. Lập trình viên không cần viết code để sử dụng các công nghệ chuyên biệt như GDI+ hay Direct3D.

## 1.2. Khả năng làm việc giữa người thiết kế giao diện và lập trình viên.

Trên thực tế để thiết kế một giao diện đẹp thì một bộ phận thiết kế sẽ đảm nhận riêng biệt. Người thiết kế giao diện sẽ dùng những phần mềm thiết kế đồ họa để tạo ra những ảnh tĩnh. Sau đó những ảnh này được chuyển tới lập trình viên để tạo ra mã trình để thực hiện hóa giao diện đã thiết kế. Hạn chế về công nghệ, sức ép tiến độ , hoặc đơn giản là hiểu nhầm ý nhau cũng sẽ làm cho người lập trình viên không hiểu được yêu cầu của thiết kế.

Để giải quyết vấn đề này WPF đã đưa ra ngôn ngữ đặc tả eXtensible Application Markup Language (XAML). Việc cho phép người thiết kế và lập trình viên làm việc chung như vậy sẽ hạn chế những lỗi phát sinh khi hiện thực hóa giao diện từ thiết kế. Thêm vào đó, nó còn cho phép hai nhóm công tác này làm việc song song, khiến mỗi bước lặp trong quy trình phát triển phần mềm ngắn đi và việc phản hồi được tốt hơn. Vì cả hai môi trường đều có khả năng hiểu và sử dụng XAML, ứng dụng WPF có thể chuyển qua lại giữa hai môi trường phát triển để sửa đổi hay bổ sung giao diện.

## 1.3. Các đặc điểm tạo ra sự khác biệt của WPF

Đồ họa phong phú: Thay vì làm việc với những điểm ảnh, được có thể làm việc trực tiếp với những đối tượng hình học cơ bản: hình chữ nhật, hình ellipese. Cũng có những đặc điểm mới như điều khiển độ trong suốt, độ mờ, cùng các hiệu ứng 3D.

Hiển thị văn bản linh hoạt: WPF có khả năng cung cấp những khả năng để hiện thị văn bản một cách phong phú ở bất cứ đâu. Ta có thể kết hợp văn bản với các đối tượng khác, hay có thể sử dụng các đặc điểm mới để hiện thị một lượng lớn văn bản một cách dễ đọc nhất.

Các hiệu ứng Animation: Ta có thể sử dụng bộ tính thời gian timer để vẽ lại hình, nhưng với WPF đặc điểm đã được tích hợp thành một phần của Framework, từ đó ta có thể định nghĩa những hiệu ứng chuyển động cho các đối tượng đồ họa khác nhau.

Hỗ trợ video, audio: Không giống như các công nghệ trước đó như Winform có sự hạn chế trong việc hỗ trợ để chạy các file Audio và Video. WPF hỗ trợ chạy tất cả các file mà Window Meida Player có thể đọc được và cho phép có thể chạy đồng thời một hoặc nhiều file. Đặc biệt WPF cung cấp các tool cho phép bạn tích hợp các nội dung video vào giao diện và cả các hiệu ứng 3D (ví dụ hiển thị video trên các mặt của hình hộp 3D).

Độ phân giải độc lập: cũng là một trong những đặc điểm khác tạo ra sự khác biệt cho công nghệ WPF. Nếu như các giao diện được thiết kế bằng các công nghệ trước như Winform vốn dựa trên GDI/GDI+ sẽ gặp nhiều phiền phức khi làm việc với những màn hình có độ phân giải khác nhau, hình vẽ sẽ lớn hơn với các màn hình có độ phân giải thấp hay nhỏ đi với khi độ phân giải màn hình cao. Nhưng trong WPF điều đó không còn là vấn đề quan ngại với những người thiết kế , bởi WPF cho phép hiển thị các đối tượng với đúng kích cỡ khi thiết kế dưới các độ phân giải khác nhau của màn hình.WPF sử dụng chính DirectX làm nền tảng cho bất cứ giao diện nào được tạo, hiển thị text thì nó cũng được tạo thông qua DirectX. Điều đó đồng nghĩa với việc những ứng dụng thông thường cũng có thể có được những hiệu ứng đồ họa phực tạp, ví dụ như các hiệu ứng trong suốt, đổ bóng và cũng nhờ thế mà có thể tăng tốc xử lý đồ họa thông qua sự tương tác giữa DirectX và card màn hình.

Hiển thị ảnh vector thay vì ảnh bitmap mang lại cho công nghệ WPF khả năng hiện thị linh hoạt với nhiều kích thước khác nhau mà không lo làm “vỡ” hình ảnh những đối tượng đồ họa.

Sự độc lập giữa thiết kế và lập trình là một trong những bước tiến mà WPF mang lại. Các đối tượng đồ họa trong WPF được thể hiện thông qua một loại mã đơn giản là XAML . Đó là một loại mã đơn giản gần giống như XML dùng để tạo và tinh chỉnh các đối tượng đồ họa. Đó là một bước tiến lớn trong quá trình thiết kế và lập trình giao diện.

# 2. Đồ họa 2D trong WPF

## 2.1. Các đối tượng hình học cơ bản-Shape

Đoạn thẳng (Line): Đoạn thẳng là một đối tượng được định nghĩa dựa trên hai đầu mút là hai điểm. Chúng ta có thể định nghĩa độ dày của đoạn thẳng, màu sắc hay cách vẽ đoạn thẳng (nét liền, nét đứt..).

Chuỗi đoạn thẳng (Polyline): Polyline là đối tượng bao gồm nhiều đoạn thẳng liên tiếp nối với nhau. Một Polyline gồm N đoạn thẳng thì được định nghĩa bới N+1 điểm.

Hình chữ nhật (Rectangle): Đối tượng Rectangle được xác định bởi tọa độ của góc trên trái và độ rộng, độ cao của hình chữ nhật cần hiển thị. Ngoài ra, ta có thể thiết lập các thuộc tính cho đường viền (màu sắc, độ dày, kiểu dáng) và tô phần bên trong của hình.

Hình elip (Ellipse) và hình tròn (Circle): Hình Ellipse được xác định bởi tọa độ của góc trên trái và độ rộng, độ cao của hình chữ nhật ngoại tiếp của Ellipse cần hiển thị.

Hình tròn là hình Ellipse với chiều rộng và chiều cao bằng nhau. Ellipse cũng có các thuộc tính cho đường viền (màu sắc, độ dày, kiểu dáng) và tô phần bên trong của hình tương tự như hình chữ nhật.

Đa giác (Polygon): Polygon là đối tượng dùng để trình diễn các hình dạng phức tạp, gồm đoạn thẳng nối tiếp khép kín. Một Polygon N đỉnh được định nghĩa bởi một tập hợp N cặp tọa độ tương ứng với mỗi đỉnh của nó.

Đường cong Bezier bằng đối tượng Path: Đối tượng Path được sử dụng để tạo nên những hình phức tạp, gồm nhiều phần nối với nhau.

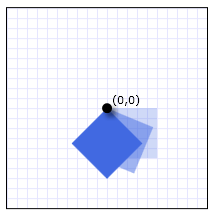
## 2.2. Biến đổi hình học – Transfrom

WPF cung cấp một số lớp (class) để hỗ trợ cho công việc biến đổi hình học:

|  |  |  |
| --- | --- | --- |
| **Lớp (Class)** | **Mô tả** | **Minh họa** |
| RotateTransform | Quay đối tượng theo góc chỉ định bởi thuộc tính Angle. | Rotate illustration |
| ScaleTransform | Co dãn đối tượng theo chiều ngang và dọc với các thuộc tính ScaleX và ScaleY. | Scale illustration |
| SkewTransform | Làm lênh hình theo các góc chỉ định bởi AngleX và AngleY. | Skew illustration |
| TranslateTransform | Dịch chuyển đối tượng tới vị trí chỉ định bởi X và Y. | Translate illustration |

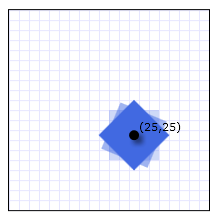
Khi biến đổi một đối tượng, ta không chỉ biến đổi bản thân nó mà còn biến đổi cả không gian tọa độ chứa đối tượng đó. Mặc định, tâm của phép biến đổi thực hiện tại gốc của hệ tọa độ của đối tượng.

Ví dụ sau minh họa sử dụng phép quay để xoay một hình chữ nhật một góc 45 độ xung quanh tâm (0,0) của trục tọa độ của đối tượng (chính là góc trên trái của hình chữ nhật), hình 2.1.



Hình 2.1: Xoay một hình chữ nhật một góc 45 độ, tâm ở đỉnh hình chữ nhật.

Ví dụ tiếp theo minh họa xoay hình chữ nhật một góc 45 độ quanh tâm của chính nó, có tọa độ là (25,25).

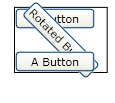


Hình 2.2 : Sử dụng phép quay để xoay một hình chữ nhật một góc 45 độ.

Không chỉ dừng ở việc biến đổi các đối tượng hình học, WPF còn cho phép biến đổi các điều khiển (Control).

Ví dụ sau minh họa xoay một nút bấm một góc 45 độ theo chiều kim đồng hồ, tâm góc quay là tâm của nút bấm. Mặc định thì tâm góc quay là góc trên trái của nút bấm. Sử dụng thuộc tính RenderTransformOrigin của Button và thiết lập ví trí tâm quay. Cặp giá trị của RenderTransformOrigin là số thực từ 0 đến 1, được tính theo tỷ lệ chiều rộng và cao của đối tượng tính từ góc trên trái.

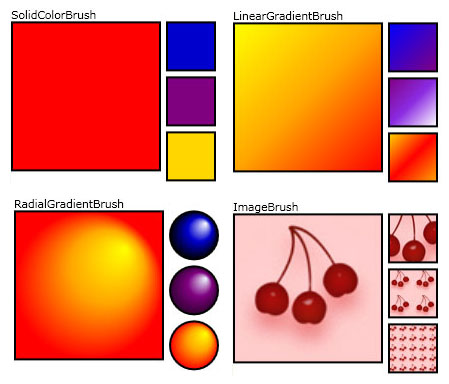
Sử dụng thuộc tính RenderTransform của Button và để thực hiện biến đổi hình học, góc quay được thiết lập nhờ thuộc tính Angle của đối tượng RotateTransform, hình 2.3



Hình 2.3 : Hình ví dụ thuộc tính RenderTransform và RotateTransform.

# 3. Sử dụng chổi tô – Brush

Tất cả những gì chúng nhìn thấy trên màn hình, chúng hiển thị được là nhờ được tô bởi chối tô (Brush). Chối tô có thể sử dụng để tô nền của một nút bấm (Button), tô các nét chữ (Text) hay tô màu bên trong cho một đối tượng hình học như hình chữ nhật, đa giác... Trong phần này chúng ta sẽ tìm hiểu cách sử dụng chổi tô trong WPF bằng mã lệnh XAML để tô các đối tượng hình học theo nhiều cách khác nhau như tô màu đồng nhất (Solid Color), tô kiểu đổ màu theo tuyến tính (Linear Gradient Color), tô đổ màu dọc theo bán kính hình tròn (Radial Gradient Color) và sử dụng ảnh bitmap để tô.



Tô đổ màu theo tuyến tính

Tô màu đồng nhất

Tô bằng ảnh Bitmap

Tô đổ màu theo tâm tròn

Hình 3.1 : Minh họa một số kiểu tô hình

Thao tác cơ bản để tô vẽ cho một đối tượng trước hết là tạo một đối tượng chổ tô (Brush) tùy theo từng loại như trên, sau đó gắn chổi tô với thuộc tính có liên quan của đối tượng cần sử dụng chổi tô này. Mỗi loại đối tượng có một số thuộc tính khác nhau để chỉ định tô màu cho phần bên trong của nó. Bảng sau đây liệt kê một số loại đối tượng và thuộc tính được dùng để gắn với chổi tô.

|  |  |
| --- | --- |
| **Loại đối tượng (Class)** | **Thuộc tính tô vẽ (Brush properties)** |
| Border | BorderBrush, Background |
| Control | Background, Foreground |
| Panel | Background |
| Pen | Brush |
| Shape | Fill, Stroke |
| TextBlock | Background |

Bảng 3.1: Một số loại đối tượng và thuộc tính được dùng để gắn với chổi tô.

## 3.1. Tô màu đồng nhất – Solid Color

Có nhiều cách để tô màu đồng nhất cho một đối tượng, ta có thể sử dụng trực tiếp thuộc tính tô màu của đối tượng (đối tượng hình học sử dụng thuộc tính Fill, đối tương Button sử dụng thuộc tính Background,..) hoặc tạo đối tượng tên là SolidColorBrush để định nghĩa màu sắc cần tô và gắn cho đối tượng cần sử dụng chổi tô đồng nhất này.

Đối tượng SolidColorBrush sử dụng thuộc tính Color để chỉ định màu cần tô. Giá trị gắn cho Color có thể là tên của mầu đã được định nghĩa sẵn (là các thuộc tính tĩnh của đối tượng Brush) như Red, MediumBlue... hoặc sử dụng công thức biểu diễn màu theo dạng mẫu 32 bit “*#RRGGBB*”, trong đó RR (giá trị của màu đỏ - Red), GG (giá trị của màu lá cây - Green), BB (giá trị của màu da trời – Blue) là các số Hexa (hệ đếm cơ số 16) có giá trị nằm trong khoảng từ 00 đến FF ( tương đương với giá trị từ 0 đến 255) . Ngoài ra, có thẻ sử dụng mẫu *“#AARRGGBB”*, AA là độ Alpha dùng để chỉ độ trong suốt của màu. Độ Alpha càng gần với 0 (00 hệ 16) thì màu càng trở nên trong suốt và càn gần với 255 (FF hệ 16) thì mầu càng rõ nét.

Sau đây là ví dụ hiển thị một hình chữ nhật được tô đồng nhất màu đỏ.



Ta cũng có thể tô màu cho các điều khiển nhờ thuộc tính Background. Ví dụ, tô màu cho nút bấm

## 3.2. Tô màu theo định hướng tuyến tính – Linear Gradient Color

Tô đổ màu (hay tản màu) là kỹ thuật tô một vùng bằng nhiều màu trộn với nhau dọc theo một trục định hướng (Gradient axis). Bạn có thể sử dụng kỹ thuật này để tạo nên các hình ảnh ấn tượng với độ sáng hay độ bóng của màu tạo cảm giác ba chiều (3D) hoặc cũng có thể dùng kỹ thuật này để tạo ra các giả lập bề mặt kính, nhôm, nước hay các bề mặt vật liệu mềm khác. WPF cung cấp hai loại đổ màu là đổ màu theo hướng tuyến tính (đối tượng LinearGradientBrush) hay đổ màu dọc theo bán kính hình tròn (đối tượng RadialGradientBrush).

Đối tượng LinearGradientBrush dùng để tô một vùng theo kỹ thuật tản màu dựa trên kỹ thuật nội suy các màu nằm giữa các cặp màu chỉ định dọc theo một trục (Gradient axis) dạng đường thẳng. Hướng của đường thẳng chính là hướng đổ màu, bạn phải chỉ định những màu cần xuất hiện tại các điểm nằm trên đường thẳng này nhờ đối tượng GradientStop. Ta có thể chỉ định hướng đổ màu nằm ngang từ trái qua phải, nằm dọc từ trên xuống, hay theo đường chéo bất kỳ. Mặc định thì hướng đổ màu sẽ theo đường chéo từ góc trên bên trái tới góc dưới bên phải.

Dưới đây là một ví dụ vẽ một hình chữ nhật gồm bốn màu khác nhau được tô theo chế độ trộn màu.

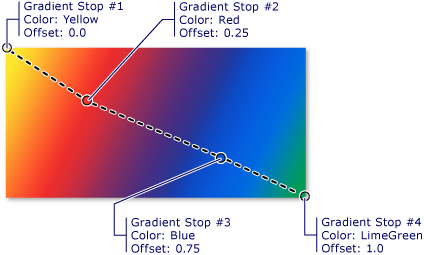


Hình 3.2 : Tô theo hướng tuyến tính

Thuộc tính StartPoint và EndPoint của LinearGradientBrush để xác định điểm đầu và điểm của của trục tô mỗi thuộc tính này được xác định bởi một cặp giá trị “x,y” . Trong đó, x và y là các số thực (double) có giá trị từ 0 đến 1, tương ứng với 0% và 100% tính tương đối so với đỉnh trên trái của hình chữ nhật cần vẽ (hay đỉnh của hình chữ nhật bao đối với các hình khác như Ellipse, Polygon,..).

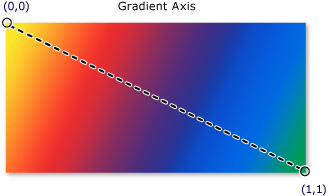
Thẻ <GradientStop Color="Color value" Offset="m.n" /> để chỉ định các điểm chốt nằm dọc theo trục tô. Thuộc tính Color xác định màu cần tô tại điểm chốt. Thuộc tính Offset nhằm xác định vị trí của điểm chốt, giá trị này là một số thực nằm trong khoảng từ 0 đến 1, giá trị càng gần 0 thì càng gần với điểm xuất phát StartPoint, giá trị gần với 1 thì càng gần với điểm kết thúc EndPoint của trục tô. Hệ thống sẽ tự động nội suy các màu nằm giữa các cặp điểm chốt.

Hình dưới đây minh họa các điểm chốt của ví dụ trên. Đường nét đứt chỉnh là trục tô (Gradient Axis) còn vòng tròn là vị trí của điểm chốt (GradientStop).



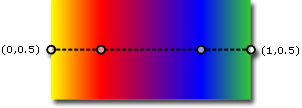
Hình 3.3 : Minh họa các điểm chốt

Tọa độ các điểm StartPoint và EndPoint của trục tô ảnh hưởng đến hướng tô màu. Mặc định thì giá trị của StartPoint là (0,0) và EndPoint là (1,1), nghĩa là trục tô là đường chéo của hình chữ nhật.



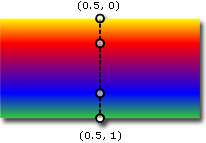
Hình 3.4: Trục tô mặc định của tô màu tuyến tính.

Hình 3.5 dưới đây minh họa các tô đổ màu theo chiều ngang gồm bốn màu:



Hình 3.5 : Trục tô màu theo chiều ngang.

Hình 3.6, tô hình chữ nhật theo chiều đứng:

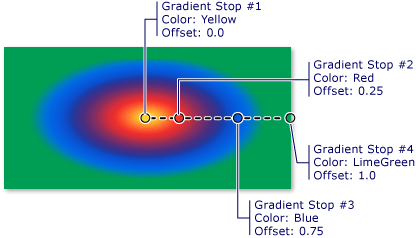


Hình 3.6 : Trục tô màu theo chiều đứng.

## 3.3. Đổ màu theo bán kinh hình tròn – Radial Gradient

Kỹ thuật tô đổ màu theo bán kính hình tròn tương tự như kỹ thuật đổ màu tuyến tính, nhưng điểm xuất phát bắt đầu từ tâm đường tròn và màu được lan dần ra ngoài cho tới biên của đường tròn, sử dụng đối tượng tên là RadialGradientBrush. Các điểm chốt vẫn sử dụng đối tượng GradientStop như phần trên.

Dưới đây là ví dụ minh họa cách tô đổ màu theo bán kính hình tròn



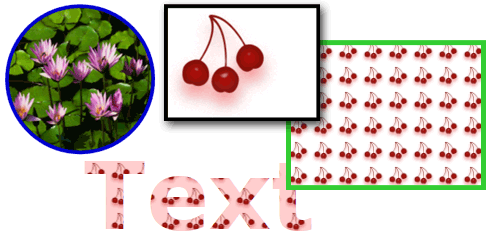
Hình 3.7 : Hình minh họa cách tô đổ màu theo bán kính hình tròn.

Các thông số Center - Tọa độ tâm đường tròn, RadiusX - Bán kính ngang, RadiusY - Bán kính dọc được nhật các giá trí số thực trong khoảng từ 0 đến 1 tương ứng với tỷ lệ khoảng cách tới đỉnh trên trái của hình chữ nhật.

Thông số GradientOrigin là tọa độ của điểm xuất phát. Các điểm chốt sẽ chạy từ điểm GradientOrigin này, dọc theo đường kính hình tròn.

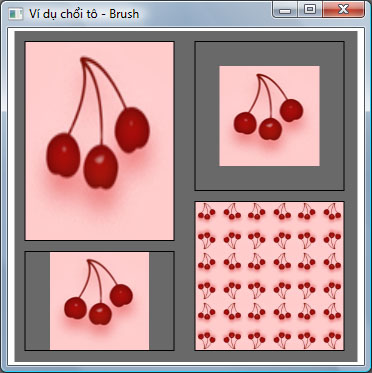
## 3.4. Tô màu bằng Bitmap

Ngoài các kỹ thuật tô màu cho các đối tượng hình học như trên, WPF còn hỗ trợ tô một vùng bằng những hình ảnh có sẵn (ảnh Bitmap, JPG,..) một cách dễ dàng nhờ đối tượng ImageBrush. Ta chỉ cần đưa tệp ảnh vào tài nguyên của Project và gắn đường dẫn anh cho thuộc tính ImageSource của đối tượng ImageBrush, sau đó dùng chổi tô ImageBush để tô các đối tượng hình học (Shape), các điều khiển (Control), Panel hay Text….



Hình 3.8 : Các mẫu (pattern)

Dưới đây là ví dụ minh họa tô màu cho hình chữ nhật bằng hình ảnh.



Ảnh tự động co dãn phủ kín hình chữ nhật

Ảnh giữ nguyên kích thước gốc

Ảnh co dãn nhưng giữ nguyên tỷ lệ

Sử dụng Viewport và cách tô kiểu xếp lợp

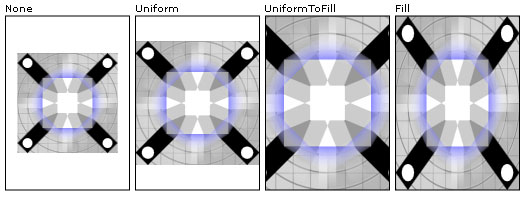
Hình 3.9 . Minh họa tô một vùng bằng hình ảnh

Thuộc tính ImageSource của ImageBrush để chỉ định đường dẫn đến tệp hình ảnh.

Thuộc tính Stretch để chỉ định các co dãn hình khi tô vùng, giá trị mặc định của Stretch là Fill. Thuộc tính này có các giá trị:

* None: Chổi tô không tự động co dãn hình.
* Uniform: Chổi tô co dãn hình trùng khít với một chiều của vùng tô nhưng giữ nguyên tỷ lệ của ảnh gốc.
* UniformToFill: Chổi tô co dãn hình phủ kín vùng tô nhưng giữ nguyên tỷ lệ của ảnh gốc.
* Fill: Chổi tô co dãn hình phủ kín vùng tô, không giữ tỷ lệ ảnh. Nếu tỷ lệ hai chiều của vùng tô khác với tỷ lệ hai chiều của ảnh thỉ ảnh tô sẽ bị méo.

Các giá trị thuộc tính Stretch được minh họa như hình 3.10 dưới đây:

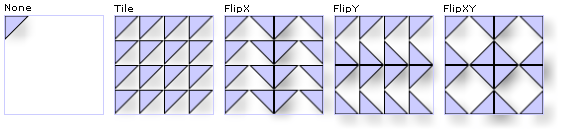


Hình 3.10 : Các kết quả của giá trị thuộc tính Stretch.

Thuộc tính TileMode chỉ định kiểu xếp lợp, gồm các giá trị sau:

* None: Không xếp lợp.
* Tile: Xếp lợp để phủ kín vùng cần tô.
* FlipX: Xếp lợp để phủ kín vùng cần tô, hình được lật theo chiều ngang.
* FlipY: Xếp lợp để phủ kín vùng cần tô, hình được lật theo chiều dọc.
* FlipXY: Xếp lợp để phủ kín vùng cần tô, hình được lật theo cả hai chiều.

Các thuộc tình trên được minh họa bởi hình 3.11 dưới:

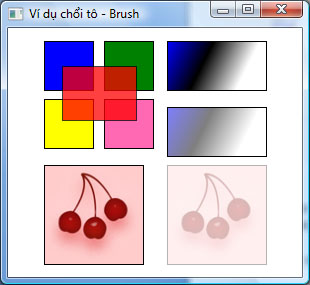


Hình 3.11: Các kết quả của giá trị thuộc tính TileMode.

## 3.5. Thiết lập độ mờ của chổi tô – Opacity

Để tạo độ mờ (Opacity), WPF cung cấp thuộc tính Opacity áp dụng cho các đối tượng hình học (Shape), hình ảnh hay RadientBrush...

Ví dụ sau đây minh họa thiết lập độ mờ cho chổi tô:



Chổi tô Image với độ mờ là 0.3

Chổi tô Gradient với độ mờ là 0.5

Chữ nhật màu đỏ được tô với độ mờ là 0.75

Chữ nhật màu đỏ được tô với độ mờ là 0.75

Chổi tô Gradient với độ mờ là 0.5

Chổi tô Image với độ mờ là 0.3

Hình 3.12 : Hình ví dụ chổi tô với độ mờ

Giá trị của Opacity là một số thực từ 0 đến 1. Nếu Opacity càng gần bằng 0 thì chổi tô là trong suốt, giá trị càng gần 1 thị chổi tô càng rõ nét.

Tất cả những gì chúng nhìn thấy trên màn hình, chúng hiển thị được là nhờ được tô bởi chối tô (Brush). Chối tô có thể sử dụng để tô nền của một nút bấm (Button), tô các nét chữ (Text) hay tô màu bên trong cho một đối tượng hình học như hình chữ nhật, đa giác…Thao tác cơ bản để tô vẽ cho một đối tượng trước hết là tạo một đối tượng chổ tô (Brush) tùy theo từng loại như trên, sau đó gắn chổi tô với thuộc tính có liên quan của đối tượng cần sử dụng chổi tô này. Mỗi loại đối tượng có một số thuộc tính khác nhau để chỉ định tô màu cho phần bên trong của nó. Bảng sau đây liệt kê một số loại đối tượng và thuộc tính được dùng để gắn với chổi tô.

|  |  |
| --- | --- |
| **Loại đối tượng (Class)** | **Thuộc tính tô vẽ (Brush properties)** |
| Border | BorderBrush, Background |
| Control | Background, Foreground |
| Panel | Background |
| Pen | Brush |
| Shape | Fill, Stroke |
| TextBlock | Background |

# 4. Animation

Animation cho phép tạo giao diện người dùng (User Interface) động một cách thực thụ. Điều này thường được thể hiện thông qua các hiệu ứng, ví dụ như các icon sẽ nhô lên khi lướt chuột qua, các logo xoay tròn, các đoạn text cuộn trong khi xem và còn rất nhiều nữa. Đôi khi việc sử dụng các hiệu ứng này làm cho ứng dụng trông có vẻ hào nhoáng quá mức và không cần thiết. Nhưng nếu sử dụng hợp lý, animation sẽ nâng cao tầm ứng dụng . Làm cho ứng dụng dễ gần gũi hơn, tự nhiên hơn và trực quan hơn (ví dụ như các button di chuyển khi di chuột tới). Animation có thể tạo những chú ý với những thành phần quan trọng và hướng dẫn người dùng chuyển tới các trang mới.

Animation là một phần của WPF. Điều này có nghĩa là không cần phải sử dụng bộ đếm (Timers) và những sự kiện được gõ bằng tay để tạo các hiệu ứng. Thay vào đó có thể tạo và cấu hình chúng bằng cách sử dụng các lớp và đặt các hiệu ứng vào bằng mã lệnh C#.

Trong chương này, chúng ta sẽ tìm hiểu các lớp animation mà WPF cung cấp. Bạn sẽ thấy như thế nào để sử dụng chúng trong code và như thế nào để xây dựng và quản lý chúng trong XAML. Song song sẽ là các ví dụ về animation, bao gồm các hình ảnh với hiệu ứng mờ, các button xoay tròn và các thành phần co giãn.

## 4.1. Basic Animation

Hãy xét đến những nguyên tắc đầu tiên của animation trong WPF, mỗi một hành động trong animation là một thuộc tính độc lập đơn lẻ. Tuy nhiên tất cả đều có giới hạn. Để sử dụng các thuộc tính tạo hiệu ứng cần phải có những lớp hỗ trợ loại dữ liệu (data types) đó. Ví dụ ở đây như Button, chiều rộng của Button sử dụng loại dữ liệu là double và để tạo hiệu ứng cho Button cần sử dụng lớp DoubleAnimation. Tuy vậy, Button.Padding lại sử dụng cấu trúc Thinkness, vì thế nó đòi hỏi lớp ThinknessAnimation.

Có rất nhiều loại dữ liệu (data types) không có lớp animation tương ứng bởi vì animation không thể thực hiện được. Ví dụ dễ thấy nhất ở đây là bản liệt kê (enumerations). Bạn có thể quản lý được các thành phần đặt trong một layout panel bằng cách sử dụng thuộc tính HorizontalAlignment và đặt giá trị từ bản liệt kê các giá trị của thuộc tính HorizontalAlignment. Tuy nhiên thuộc tính HorizontalAlignment chỉ cho phép chọn bốn giá trị (Left, Right, Center, và Stretch), đây là giới hạn thật sự trong việc sử dụng animation. Mặc dù bạn có thể chuyển đổi các hướng khác nhau nhưng nó không thể mềm mại được (Smoothly) và đó là lý do không có hiệu ứng nào cho thuộc tính HorizontalAlignment. Tuy có thể tự xây dựng các hiệu ứng, nhưng sẽ vẫn không có animation với bốn giá trị như vậy.

## 4.2. Các lớp Animation

Animation có hai dạng : đó là sự khác nhau giữa giá trị bắt đầu và kết thúc (được gọi là phép nội suy - interpolation) và loại kia là thuộc tính bị thay đổi một cách đột ngột bằng các giá trị khác nhau. DoubleAnimation và ColorAnimation là một ví dụ cho loại đầu tiên, sử dụng phép nội suy để làm mượt quá trình thay đổi giá trị. Tuy nhiên phép nội suy lại không thể tạo được cảm giác mượt mà cho một số loại dữ liệu, ví dụ ở đây là chuỗi (String) và thay vì sử dụng phép nội suy, những loại dữ liệu bị thay đổi một cách đột ngột này sẽ sử dụng key frame animation.

Một vài loại dữ liệu có lớp key frame animation nhưng lại không có lớp interpolation animation. Ví dụ ở đây lại là chuỗi, bạn có thể tạo hiệu ứng cho chuỗi thông qua key frame animation nhưng với interpolation animation thì không được. Có thể thấy là mọi loại dữ liệu đều hỗ trợ key frame animation, trừ khi là chúng không hỗ trợ animation.

Sau đây là các lớp animation, ở đây bao gồm 42 lớp:

* BooleanAnimationUsingKeyFrames
* ByteAnimation
* ByteAnimationUsingKeyFrames
* CharAnimationUsingKeyFrames
* ColorAnimation
* ColorAnimationUsingKeyFrames
* DecimalAnimation
* DecimalAnimationUsingKeyFrames
* DoubleAnimation
* DoubleAnimationUsingKeyFrames
* DoubleAnimationUsingPath
* Int16Animation
* Int16AnimationUsingKeyFrames
* Int32Animation
* Int32AnimationUsingKeyFrames
* Int64Animation
* Int64AnimationUsingKeyFrames
* MatrixAnimationUsingKeyFrames
* MatrixAnimationUsingPath
* ObjectAnimationUsingKeyFrames
* PointAnimation
* PointAnimationUsingKeyFrames
* PointAnimationUsingPath
* Point3DAnimation
* Point3DAnimationUsingKeyFrames
* QuarternionAnimation
* QuarternionAnimationUsingKeyFrames
* RectAnimation
* RectAnimationUsingKeyFrames
* Rotation3DAnimation
* Rotation3DAnimationUsingKeyFrames
* SingleAnimation
* SingleAnimationUsingKeyFrames
* SizeAnimation
* SizeAnimationUsingKeyFrames
* StringAnimationUsingKeyFrames
* ThicknessAnimation
* ThicknessAnimationUsingKeyFrames
* VectorAnimation
* VectorAnimationUsingKeyFrames
* Vector3DAnimation
* Vector3DanimationUsingKeyFrames

## 4.3. Animation trong code

Ở đây chắc bạn sẽ tự hỏi WPF đã xác định sự thay đổi khi sử dụng phép nội suy như thế nào, thật may mắn vì điều này hoàn toàn tự động và thay đổi này được thể hiện qua từng frame, ở đây số frame WPF sử dụng là 60 frame trên giây và vì thế cứ 1/60 s thì WPF sẽ tính toán để thay đổi giá trị tương ứng cho thuộc tính.

Cách tốt nhất để sử dụng animation là sử dụng các lớp animation, để cấu hình và thay đổi thuộc tính và sau đó sử dụng BeginAnimation() cho thành phần muốn thay đổi.

Ví dụ đơn giản sau đây sẽ thể hiện một animation cơ bản, đó là tăng kích thước của Button khi click chuột vào Button đó, WPF sẽ kéo dài Button một cách mượt mà:





Hình 4.1 : Button với animation đơn giản

Qua ví dụ này ta nắm bắt được một điều, để hoạt động được, các thuộc tính mà bạn muốn trình diễn phải được thiết lập trước đó. Ở đây có ba chi tiết cần chú ý tới khi tạo animation sử dụng phép nội suy (interpolation): đó là giá trị bắt đầu (From), giá trị kết thúc (To) và thời gian diễn ra animation đó (Duration). Với ví dụ này thì giá trị kết thúc của chiều rộng của Button phụ thuộc vào chiều rộng của cửa sổ Window. Các thuộc tính vừa nêu trên được thuộc về lớp animation interpolation.

* From

Giá trị From là giá trị bắt đầu cho thuộc tính chiều rộng. Nếu nhấn chuột vào Button nhiều lần thì mỗi lần nhấp chuột chiều rộng sẽ được thiết lập là 160, và sau đó animation được gọi.

Xét ví dụ trên nhưng sẽ không thiết lập thuộc tính From, khi đó cho dù có bấm nhiều lần thi Button cũng không có hiệu ứng nữa vì chiều rộng của Button đã bằng chiều rộng của nó khi kết thúc.

DoubleAnimation widthAnimation = new DoubleAnimation();

widthAnimation.To = this.Width - 30;

widthAnimation.Duration = TimeSpan.FromSeconds(5);

button.BeginAnimation(Button.WidthProperty, widthAnimation);

Có một điều cần lưu ý là khi không sử dụng From thì Button trên phải được thiết lập chiều rộng nếu không sẽ sinh ra lỗi.

* To

Có thể bỏ giá trị From và cũng có thể bỏ giá trị To. Trong trường hợp này, có thể bỏ cả 2 giá trị là From và To để tạo animation như sau:

DoubleAnimation widthAnimation = new DoubleAnimation();

widthAnimation.Duration = TimeSpan.FromSeconds(5);

button.BeginAnimation(Button.WidthProperty, widthAnimation);

Thoạt nhìn, hiệu ứng Button được kéo dài ra không có, đó chính là do giá trị From và To đã không được thiết lập, có nghĩa là giá trị From và To là giống nhau. Animation vẫn được thực hiện nhưng ta lại không thấy các hiệu ứng đó.

* By

Thay vì sử dụng To, có thể sử dụng thuộc tính By. Thuộc tính By được dùng để tạo animation với khoảng thay đổi được định trước, có thể hiểu ở đây là khoảng thay đổi giữa From và To. Ví dụ, tạo animation cho Button là tăng 10 đơn vị chiều rộng khi nhấn vào Button đó:

DoubleAnimation widthAnimation = new DoubleAnimation();

widthAnimation.By = 10;

widthAnimation.Duration = TimeSpan.FromSeconds(0.5);

button.BeginAnimation(Button.WidthProperty, widthAnimation);

Có thể thay bằng thuộc tính To cho thuộc tính By như sau:

button.To = cmdGrowIncrementally.Width + 10;

* Duration

Thuộc tính Duration lấy một khoảng thời gian (milli giây, giây, phút, giờ hay bất cứ khoảng thời gian nào muốn sử dụng) giữa khoảng thời gian animation bắt đầu và kết thúc.

## 4.4. Animation kép – Simultaneuos Animations

Có thể sử dụng BeginAnimation() để thực thi hơn một animation trong một thời điểm. Phương thức BeginAnimation() được trả về ngay tức khắc, điều nay cho phép thực hiện hai animation đồng thời:

DoubleAnimation widthAnimation = new DoubleAnimation();

widthAnimation.From = 160;

widthAnimation.To = this.Width - 30;

widthAnimation.Duration = TimeSpan.FromSeconds(5);

DoubleAnimation heightAnimation = new DoubleAnimation();

heightAnimation.From = 40;

heightAnimation.To = this.Height - 50;

heightAnimation.Duration = TimeSpan.FromSeconds(5);

button.BeginAnimation(Button.WidthProperty, widthAnimation);

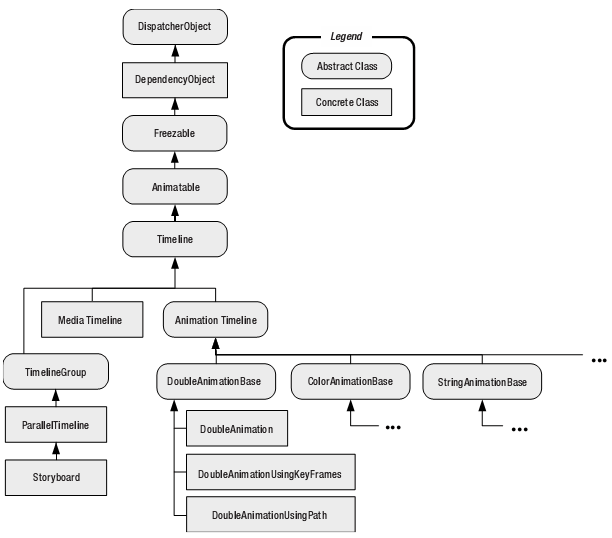
button.BeginAnimation(Button.HeightProperty, heightAnimation);

Trong ví dụ trên, hai animation được diễn ra không đồng bộ. Có nghĩa là chiều rộng và cao không tăng trong một thời điểm chính xác (chiều rộng tăng rồi mới tới chiều cao) để có thể làm đồng bộ, cần sử dụng storyboard, phần này sẽ được nêu ở phần sau của chương này.

## 4.5. The Timeline Class

Mỗi animation đều xoay quanh một vài thuộc tính. Bạn đã được thấy nhiều thuộc tính như: From, To, Duration và FillBehavior. Trước khi tiếp tục nên xem lại các thuộc tính mà bạn đã làm việc với chúng.

Hình 4.2 sẽ thể hiện cho thấy thứ bậc kế thừa của các kiểu mẫu (type) trong animation của WPF. Hình này bao gồm tất cả các lớp cơ sở và chỉ thể hiện một ít các kiểu mẫu animation.



Hình 4.2 : Hệ thống các lớp trong animation

Dưới đây là bảng 4.1, các thuộc tính TimeLine:

|  |  |
| --- | --- |
| **Tên** | **Mô tả** |
| BeginTime | Thiết lập độ trễ được thêm vào trước khi animation được bắt đầu. Độ trễ này sẽ được cộng vào tổng thời gian, ví dụ như năm giây thực hiện animation và năm giây độ trễ thì quá trình này sẽ tốn 10 giây. BeginTime là một thuộc tính rất hữu dụng khi muốn đồng bộ các animation khác nhau bắt đầu tại cùng một thời điểm. |
| Duration | Thiết lập khoảng thời gian mà animation thực hiện, từ khi bắt đầu đến khi kết thúc. |
| SpeedRatio | Tăng hay giảm tốc độ thực hiện của animation. Vì thế khi tăng SpeedRatio thì animation sẽ kết thúc nhanh hơn (ví dụ khi bạn thiết lập SpeedRatio bằng 5 thì animation sẽ kết thúc sớm hơn 5 lần) và tương tự khi giảm SpeedRatio. Có thể thay đổi Duration để có kết quả tương đương như thay đổi SpeedRatio. |
| AccelerationRatio và DecelerationRatio | Dùng để tạo animation phi tuyến (nonlinear), animation sẽ bắt đầu chậm hơn (khi tăng AccelerationRatio) hay kết thúc chậm hơn (khi tăng DecelerationRatio). Cả hai thuộc tính này đều được thiết lập với giá trị nằm trong khoảng 0 và 1. Thêm nữa cả tổng của hai giá trị này không được lớn hơn 1. |
| AutoReverse | Nếu được thiết lập là true thì animation sẽ tự động thực hiện ngược lại khi quá trình đã kết thúc, trở lại với giá trị ban đầu. Nếu tăng SpeedRatio thì cả hai quá trình thực hiện và quay lại đều thay đổi. Trong khi đó thì BeginTime lại chỉ có tác động đến quá trình thực hiện ban đầu, còn đối với quá trình ngược lại thì không có tác dụng. |
| FillBehavior | Xác định điều gì sẽ xảy ra khi animation kết thúc. Thông thường thì giá trị khi kết thúc được giữ nguyên (FillBehavior.HoldEnd) , tuy nhiên có thể chọn cho giá trị đó được quay về là giá trị ban đầu bằng việc thiết lập FillBehavior.Stop. |
| RepeatBehavior | Cho phép lặp lại quá trình animation trong một khoảng thời gian xác định hay một số lần xác định. |

Bảng 4.1: Bảng thuộc tính TimeLine

Lấy ví dụ button ở trên xét lần lượt xem các thuộc tính TimeLine thực hiện như thế nào:

DoubleAnimation widthAnimation = new DoubleAnimation();

widthAnimation.From = 160;

widthAnimation.To = this.Width - 30;

widthAnimation.Duration = TimeSpan.FromSeconds(2);

button.BeginAnimation(Button.WidthProperty, widthAnimation);

* BeginTime

widthAnimation.BeginTime = TimeSpan.FromSeconds(5);

khi bấm vào button thì sau 5 giây button mới có hiệu ứng tăng chiều rộng.

* SpeedRatio

widthAnimation.SpeedRatio = 2;

Tốc độ của animation tăng gấp đôi.

* AccelerationRatio và DecelerationRatio

widthAnimation.AccelerationRatio = 1;

Tăng thời gian bắt đầu animation.

widthAnimation.DecelerationRatio = 1;

Tăng thời gian kết thúc animation.

* AutoReverse

widthAnimation.AutoReverse = true;

Sau khi tăng chiều rộng thì quá trình giảm chiều rộng của button sẽ được thực hiện tương tự.

* FillBehavior

widthAnimation.FillBehavior = FillBehavior.Stop;

Sau khi animation kết thúc thì chiều rộng của button trở về với giá trị ban đầu.

* RepeatBehavior

widthAnimation.RepeatBehavior = new RepeatBehavior(TimeSpan.FromSeconds(10));

Vì thời gian thực hiện animation là 2 giây nên khi thiết lập RepeatBehavior là 10 giây thì animation sẽ được thực hiện 5 lần.

## 4.6. Animation và Storyboards

Như những gì đã tìm hiểu thì WPF animation được trình diễn bằng một nhóm các lớp animation và phải thiết lập các thông tin thích hợp, như là giá trị bắt đầu, giá trị kết thúc, thời gian thực hiện, sử dụng một nhóm các thuộc tính đã trình bày ở phần trước, rõ ràng là những điều này hoàn toàn có thể làm trên Xaml.

Có hai thành phần với animation trên Xaml đó chính là storyboard và trigger:

* Storyboard là thành phần của Xaml tương đương với phương thức BeginAnimation().
* Event trigger phản hồi lại khi thuộc tính bị thay đổi hay xảy ra sự kiện (ví dụ như sự kiện nhấn vào button) và điều khiển storyboard. Ví dụ như để bắt đầu animation thì event trigger phải bắt đầu storyboard.

### 4.6.1. Storyboard

Storyboard là một timeline nâng cao. Có thể sử dụng để thực hiện một nhóm các animation phức tạp, và vì thế có thể quản lý được animation như dừng lại, kết thúc, thay đổi vị trí ... Tuy nhiên đặc điểm cơ bản mà các lớp Storyboard cung cấp đó chính là khả năng trỏ tới những thuộc tính đặc biệt cũng như các thành phần đặc biệt thông qua thuộc tính TargetProperty và TargetName. Và cuối cùng storyboard sẽ khắc phục những khe hở giữa animation và các thuộc tính mà bạn muốn animation.

Dưới đây là cách định nghĩa storyboard để quản lý DoubleAnimation:

<Storyboard TargetName="cmdGrow" TargetProperty="Width">

<DoubleAnimation From="160" To="300" Duration="0:0:5"> </DoubleAnimation>

</Storyboard>

Với cú pháp trên còn có thể làm nhiều hơn nữa, bởi vì storyboard cho phép đặt rất nhiều các animation trong cùng một storyboard và mỗi animation này cho phép thực hiện trên các thuộc tính và thành phần khác nhau. Để làm được điều này cần sửa một chút so với đoạn mã trên:

<Storyboard>

<DoubleAnimation

Storyboard.TargetName="cmdGrow" Storyboard.TargetProperty="Width"

From="160" To="300" Duration="0:0:5"></DoubleAnimation>

</Storyboard>

Định nghĩa storyboard là bước đầu để tạo animation. Để thực sự thực hiện animation ta cần đến event trigger.

### 4.6.2. Event trigger

Event trigger đã được nhắc đến trong chương 8, khi tìm hiểu về style. Style cho bạn một cách để gán event trigger cho element. Tuy nhiên có định nghĩa event trigger ở bốn chỗ sau:

* Trong style – Style.Triggers
* Trong data template – DataTemplate.Triggers
* Trong control template – ControlTemplate.Triggers
* Trong element directly – FrameworkElement.Triggers

Khi tạo trigger cần xác định sự kiện nào bắt đầu trigger và những hành động (actions) nào được thực hiện bởi trigger. Với animation, hầu hết các hành động thông thường là BeginStoryboard tương đương với khi bạn gọi BeginAnimation().

Ví dụ sau đây sẽ sử dụng tập hợp Trigger của button để thêm animation vào sự kiện nhấn chuột. Khi button được nhấn, nó sẽ phình to ra.

Thuộc tính Storyboard.TargetProperty sẽ đồng nhất với thuộc tính muốn thay đổi. Nếu không chỉ rõ tên lớp, storyboard sử dụng thành phần cha, đó chính là button cần phình to ra. Nếu muốn thiết lập các thuộc tính cụ thể (ví dụ như Canvas.Left hay Canvas.Top), cần đặt toàn bộ thuộc tính đó trong dấu ngoặc như sau:

<DoubleAnimation Storyboard.TargetProperty="(Canvas.Left)" ... />

Storyboard.TargetName thì không có trong ví dụ này. Khi đó storyboard sẽ sử dụng thành phần cha đó chính là button.

Nếu muốn sử dụng chiều rộng của cửa sổ window thì cần sử dụng biểu thức data binding (data binding expression) như sau:

<DoubleAnimation Storyboard.TargetProperty="Width"

To="{Binding ElementName=window,Path=Width}" Duration="0:0:5">

</DoubleAnimation>

Với đoạn mã trên thì button sẽ phình ra đúng bằng chiều rộng của cửa sổ window.

### 4.6.3. Gắn Trigger với Style

FrameworkElement.Triggers có vẻ khác thường. Vì nó chỉ hỗ trợ event trigger. Trong khi các trigger khác (Style.Triggers, DataTemplate.Trigger, và ControlTemplate.Trigger) thì làm được nhiều điều hơn, đó là hỗ trợ ba kiểu trigger của WPF: property trigger, data trigger, và event trigger.

Sử dụng event trigger là cách thông thường để gắn animation. Tuy nhiên, đó không phải là lựa chọn duy nhất. Nếu sử dụng trigger trong style, data template, control template bạn có thể tạo thuộc tính trigger (property trigger) được tác động khi giá trị thuộc tính được thay đổi. Sau đây là một ví dụ, button sẽ trigger một storyboard khi button được nhấn vào (IsPressed).

Có thể gắn hành động tới thuộc tính trigger theo hai cách. Đó là sử dụng Trigger.EnterActions để thiết lập hành động sẽ diễn ra khi thuộc tính thay đổi (trong ví dụ trên là khi IsPressed là true) và sử dụng Trigger.ExitActions để thiết lập hành động sẽ diễn ra khi thuộc tính thay đổi ngược lại ( khi IsPressed là false).

Dưới đây là đoạn mã để gắn animation vào button:

<Button Padding="10" Name="cmdGrow" Height="40" Width="160" Style="{StaticResource GrowButtonStyle}" HorizontalAlignment="Center" VerticalAlignment="Center">

Click and Make Me Grow

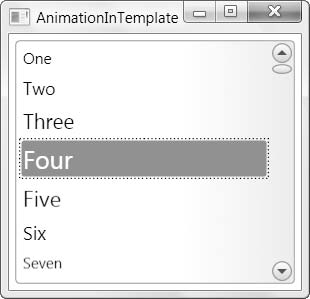
</Button>

Nên nhớ không cần phải định nghĩa style cho mỗi thành phần mà bạn sử dụng, ở đây có thể định nghĩa một lần và sử dụng cho nhiều thành phần mà bạn muốn.

### 4.6.4. Gắn Triggers với Template

Một trong những sức mạnh của WPF đó là việc sử dụng lại animation được định nghĩa trong template

Ví dụ dưới đây sẽ cho thấy một danh sách khi chuột di chuyển nhanh qua nhiều phần tử (item).



Hình 4.3: Các hiệu ứng riêng rẽ của mỗi ListBoxItem

Có thể thêm mouseover vào aniamtion (sự kiện khi chuột di chuyển tới) theo hai cách: đó là tạo một event trigger để phản hồi lại sự kiện MouseEnter và MouseLeave hay là tạo một property trigger để thêm vào hành động chuột di chuyển vào và thoát ra khi thuộc tính IsMouseOver thay đổi. Sau đây là đoạn mã ví dụ sử dụng event trigger

Trong ví dụ trên thì ListBoxItem được lớn ra tương đối chậm (vì thời gian lớn hơn một giây) và khi nhỏ lại thì nhanh hơn (0.2 giây). Tuy nhiên ở đây có 0.5 giây đỗ trễ trước khi animation bắt đầu. Có thể thấy ở đây là hiệu ứng khi di chuyển chuột vào các phần tử của listbox (ở đây là text) thì các phần tử sẽ tăng font chữ lên, và quá trình sẽ diễn ra ngược lại khi rời chuột khỏi các phần tử này.

### 4.6.5. Điều khiển các kĩ thuật phát lại – Playback

Thông thường khi thực hiện một hành động trong event trigger, Storyboard sẽ thực hiện animation. Tuy nhiên có thể sử dụng nhiều các hành động để quản lý một Storyboard được tạo. Các hành động này thuộc về lớp ControllableStoryboardAction, được liệt kê trong bảng dưới đây.

|  |  |
| --- | --- |
| **Tên** | **Mô tả** |
| PauseStoryboard | Dừng playback của animation và giữ vị trị hiện hành |
| ResumeStoryboard | Phục hồi lại playback khi animation bị dừng lại |
| StopStoryboard | Dừng playback của animation và phục hồi bộ đếm animation về giá trị ban đầu. |
| SeekStoryboard | Nhảy đến giá trị đặc biệt trong timeline của của animation. Nếu animation đang chạy, nó sẽ tiếp tục chạy ở vị trí mới. còn nếu animation đang dừng thì nó sẽ dừng ở vị trí mới và chờ để được bắt đầu. |
| SetStoryboardSpeedRatio | Thay đổi SpeedRation của toàn bộ storyboard |
| SkipStoryboardToFill | Di chuyển storyboard đến vị trí cuối cùng trong timeline |
| RemoveStoryboard | Bỏ đi animation, dừng các tiến trình animation và thuộc tính được trở về với giá trị ban đầu |

Bảng 4.2: Các hành động này thuộc về lớp ControllableStoryboardAction

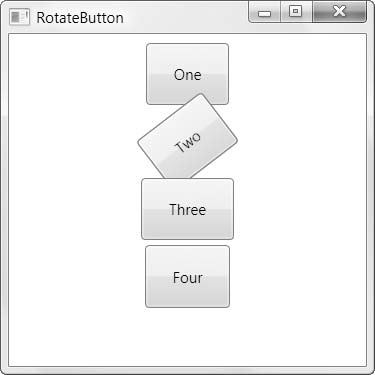
## 4.7. Kiểu Animation Revisited

Để sử dụng transform trong animation, đầu tiên phải định nghĩa tranform. Ví dụ như có thể cho các button xoay tròn thì bạn cần RotateTransform

Ở đây có event trigger để button xoay tròn khi chuột di chuyển tới, sử dụng thuộc tính là RenderTransform.Angle.

Để dừng quay cần sử dụng trigger thứ hai để phản hồi sự kiện chuột di chuyển ra khỏi button (MouseLeave)

Để button thực sự có thể quay cần thêm các trigger này vào button.



Hình 4.5 : Sử dụng transform

Sau khi tìm hiểu animation, chắc hẳn đã có thể hình dung được phần nào animation cũng như có thể áp dụng animation vào trong ứng dụng của mình một cách dễ dàng, qua đó giúp ứng dụng của mình trông gần gũi hơn, mượt mà hơn và trực quan hơn.

# 5. Kết luận

## 5.1 Kết quả đạt được

Sau một thời gian nghiên cứu công nghệ WPF và xây dựng ứng dụng minh họa, chúng em đã đạt được:

* Tìm hiểu được công nghệ tương lai của Microsoft là WPF.
* Tìm hiểu được ngôn ngữ mới là XAML.
* Hiện thực mô hình độc lập giữa lập trình viên và người thiết kế giao diện.
* Xây dựng minh họa với công nghệ WPF và sử dụng các công cụ mới như: Visual Studio 2010, Expression Blend.

## 5.2 Hạn chế của luận văn

Trong quá trình thực hiện đồ án, chúng em đã rất cố gắng nhưng sự hạn chế

về kiến thức cũng như thời gian thực hiện đồ án nên vẫn còn mắc một số các thiếu

sót và hạn chế như:

* Một số phần trong nghiên cứu công nghệ WPF còn chưa đi sâu.
* Phần ứng dụng minh họa còn đơn giản, chưa đi sâu vào thực tế.
* Chương trình minh họa cần máy tính có cấu hình mạnh.

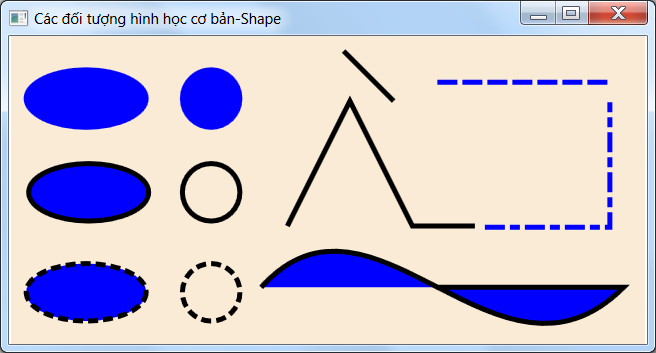
5.3. Hướng phát triển

Trước những hạn chế của luận văn, chúng em đề ra hướng khắc phục và phát triển đề tài trong tương lai như sau:

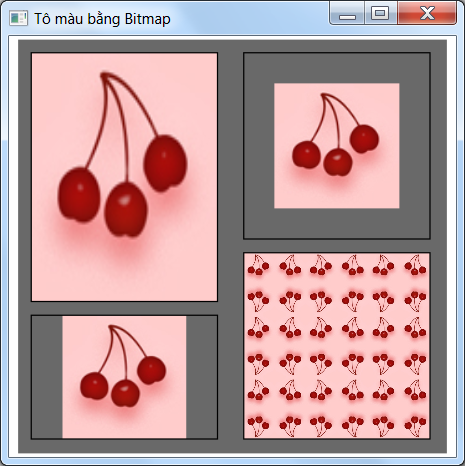
* Sẽ tiếp tục nghiên cứu sâu hơn về WPF cũng như ngôn ngữ XAML.
* Tận dụng các thế mạnh của WPF một cách triệt để hơn.
* Tiếp tục phát triển tiếp những ứng dụng minh họa cụ thể để có sử dụng trong thực tế.

# 6. Kết quả demo minh họa

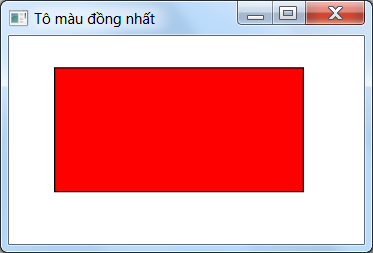
- Các đối tượng hình học cơ bản - Shape



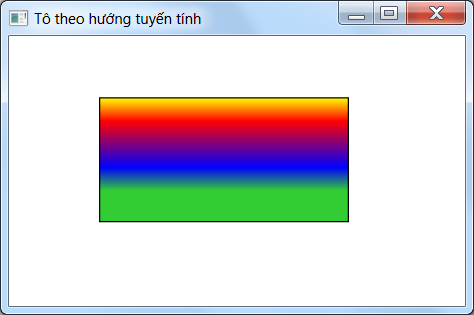
- Tô màu bằng Bitmap:



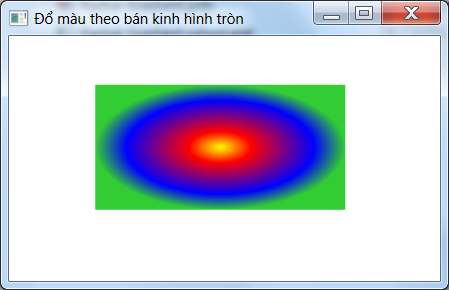
- Tô màu đồng nhất



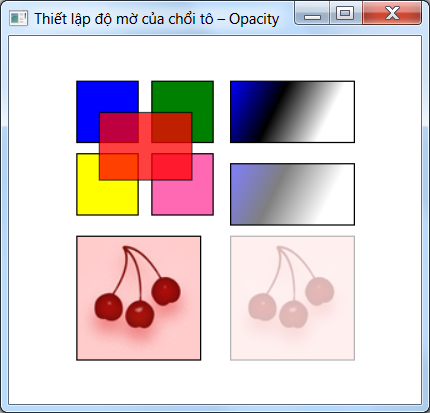
- Tô màu theo hướng tuyến tính



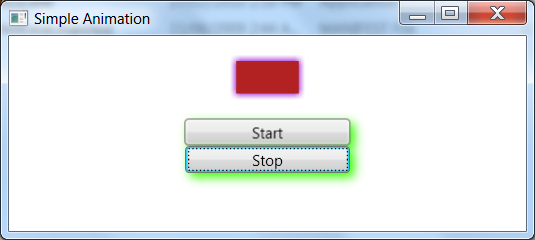
- Tô màu theo bán kính đường tròn



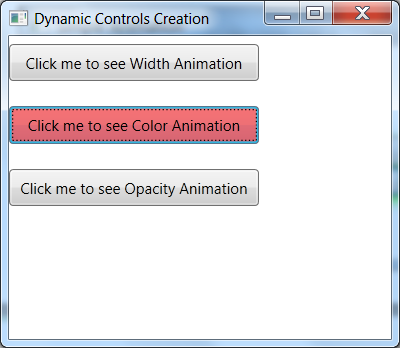
- Thiết lập độ mờ của chổi tô



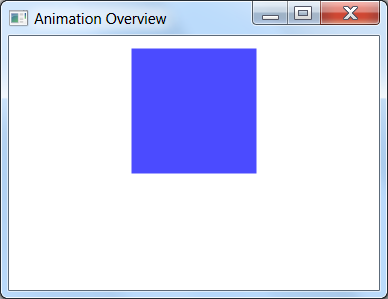
- Simple Animation



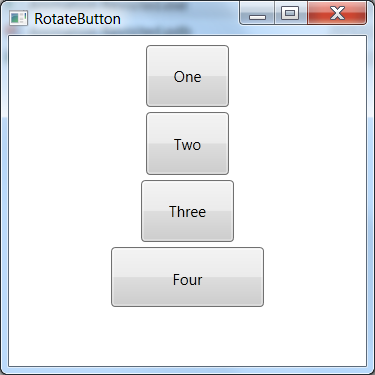
- Color Animation , Width Animation , Opactity Animation



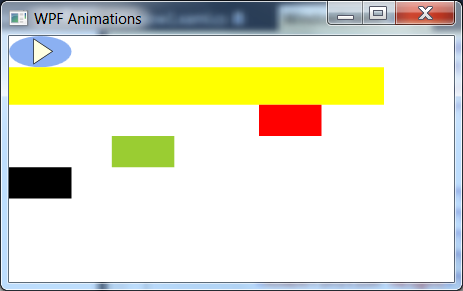
- Animation Overview



- Sử dụng transform



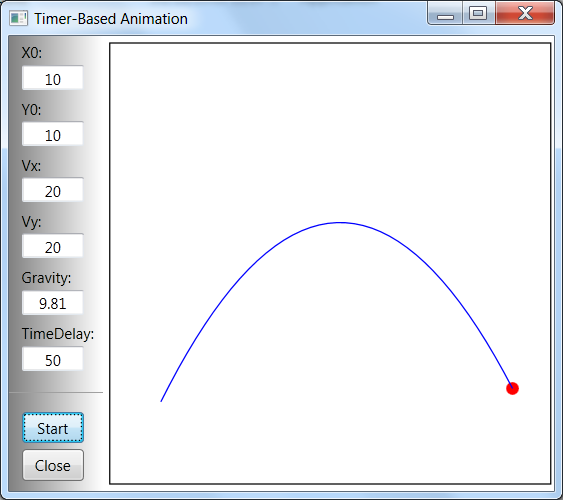
- Animations with different speeds



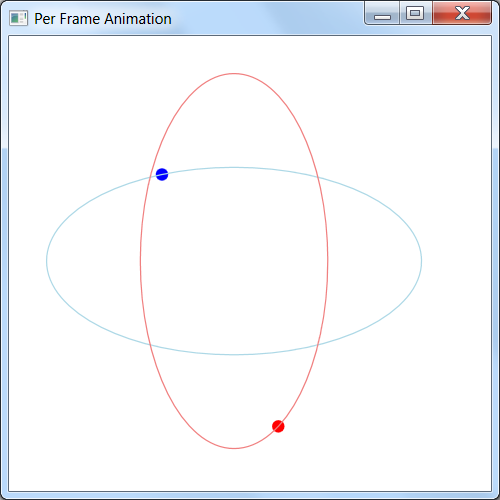
- Rolling Balls : Tạo mỗi ảnh động trong mỗi khung hình



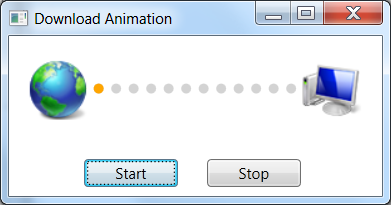
- Timer Based Animation



- Per Frame Animation



- Download Animation



**Tài Liệu Tham Khảo**

**Tiếng Việt**

* Giang Kila, 2009, WPF - 9 lessons,
* http://msdnvietnam.net/forums/p/62/79.aspx#79, MSDN Việt Nam

**Tiếng Anh**

* Professional WPF Programming (May.2007) by Chris Andrade,Shawn Livemore,Mike Meyers,Scoct Van Vliet.
* Programming Windown Presentation Foundation by Lan Griffiths,Chris Sells
* UniCAD - Practical WPF Graphics Programming (Nov 2007)
* Matthew MacDonald ,Pro WPF in C# 2010: Windows Presentation
* Matthew MacDonald ,Pro WPF in C# 2008: Windows Presentation
* Illustrated WPF by Daniel M. Solis
* MSDN Library, http://msdn.microsoft.com/en-us/library/default.aspx, Microsoft Corporation.