Human-Computer Interaction with Palimpsests

J Fishenden, Bath Spa University

ABSTRACT

This paper discusses the development of a prototype human-computer interaction (HCI) environment for the user-led exploration of time and place. It utilises earlier work from a custom HCI device to develop user interaction techniques using Kinect for Windows and the Kinect SDK. Two specific visual techniques are applied — a lens and a slider — for gesture-based manipulation of content, supported by aural commands that facilitate simplified switching between techniques. Code samples are included to illustrate the development and realisation of the techniques and their reapplication using Kinect.

Keywords: creative computing; human-computer interaction; HCI; interactive digital technologies; creative technologies; palimpsests; rich internet applications; usability testing; Kinect

INTRODUCTION

This paper describes current research in progress focused on the iterative development of creative computational techniques to explore a variety of user techniques for interacting with content related to the past of place. Work commenced with traditional HCI techniques utilising computer screens and mice; developed through the prototyping of a custom HCI device using sonar and other sensors; and is being further developed using the Windows Kinect and its associated SDK.

PALIMPSESTS AND THE PAST OF PLACE

The notion of the palimpsest has been defined as “... an overwritten manuscript; a manuscript written over a partly erased older manuscript in such a way that the old words can be read beneath the new.”2 But the concept also has a broader interpretation, encompassing the concept of multiple layers of visual or aural artefacts. This wider understanding of the palimpsest (a scope that embraces hidden layers that influence our perception and understanding of place) is the predominant focus of this research: in particular, the use of creative computational techniques that help to reveal layers of the past that lie latent beneath the present, both in vision and in sound.

Initial work explored various computational techniques for utilising a web browser-based system to enable a user to discover and navigate multi-dimensional layers of interactive visual and aural content modelled on the concept of the palimpsest. The primary conceptual model used to realise an artistic construct of the palimpsests is structured around n-tiers of visual and aural content beneath the surface level, illustrated in Figure 1 below, operating across the Cartesian co-ordinates for a three-dimensional space. Such content is typically related to visual images (still and moving) of the same place (such as a building) or other artefact (such as a map) over time. The research interest centred on developing and identifying intuitive ways for users to navigate and explore such multiple layers of content, as well as in finding methods that evoked stronger user emotional connections with the past of place.

1 j.fishenden@bathspa.ac.uk

© 2013 Bath Spa University, Centre for Creative Computing
CORE TECHNIQUES: ORIGINATION OF THE LENS AND SLIDER

LENS

The initial idea for a visualisation and interaction method for navigating the n-tiered model was that of a lens – a lens similar to a magnifying glass, but rather than magnifying content underneath the lens it would instead reveal how the particular place over which it moved had looked in the past (see Figure 2 below).

Figure 2: a lens that can "see through time"

Figure 3 (see below) illustrates this initial conceptual model, where underlying n-layers (104-106) of visual content contained in the z-plane (or axis, 110) can be revealed through the use of a circular aperture (or lens, 103) which the user (101) can freely position on the x and y axes through movements of a mouse. Wherever the lens moves in reaction to user mouse movements, it reveals the underlying visual content existent below a surface layer (102).
The technique was directed towards reacting to specific user action initiated by movement of a mouse (or another input control device, such as use of a finger on a touch screen) to navigate a circular aperture (lens) across visual content. The result is that the visualisation area over which the lens moves reveals underlying related visual content (layered in the z-axis). As the lens is moved by the user across the x and y axes of the surface layer, underlying n layers located in the z-plane are visualised in the area of the lens, revealing visual content related to those particular co-ordinates. Underlying visualisation control code tracks where the lens object is situated on-screen relative to the surface manifestation through visualisation display logic, which tracks the movements of the mouse and updates the corresponding x and y axes co-ordinates of the on-screen lens control. There are limits on these calculations so that the lens control does not wrap from one edge of the screen to the other, although such wrapping is feasible.

The initial realisation of the application of this technique was coded in Microsoft Silverlight 1.0, utilising the eXtensible Application Markup Language (XAML) for design elements and Javascript for code-behind. Silverlight’s representation of Cartesian co-ordinates for a three-dimensional space is shown in Figure 4, providing a direct correlation between theoretical and physical models.

An example early visualisation prototype is illustrated below.
Figure 5: Palimpsest Navigator Lens (without user focus)

When the user positions the cursor over the lens the underlying image becomes less opaque (see Figure 6) and an associated sound is invoked to provide aural reinforcement that the lens has entered an interactive state.

Figure 6: Palimpsest Navigator Lens (with user focus)

When the mouse button is clicked and held down, the user is able to move the lens around the screen, revealing an additional visual layer within the locus of the lens of the same place at an earlier time. The user can choose to stop moving the lens around the screen by releasing the mouse button. The cursor then disengages from the lens and can be moved away, restoring the lens to its passive state.

SLIDER

A second complementary technique for navigating visual layers of place over time was developed utilising an on-screen slider control. This was originally programmed in the same environment as the lens, Silverlight 1.0, and is illustrated in Figure 7. Movement of the slider progressively reveals an additional (previous) layer of the same place at an earlier time.

Figure 7: Palimpsest Slider Control
Unlike the lens control, the slider control affects the entire visual image displayed, also allowing fine-grained merging of the images so that the user can adjust the extent to which one or other images dominate the mix, or whether both are mixed equally (see Figure 8).

![Figure 8: Palimpsest Slider Control (images mixed)](image)

In other applications of this technique more than two layers are manipulated, a model applied to maps of the same area over time (see Figure 9).

![Figure 9: Palimpsest Slider Control (maps - images mixed)](image)

**CODING THE LENS**

The n-tier navigation model used several techniques throughout the research, one of these being the development of the lens able to “see” through layers of the same place over time. This section illustrates how the code for the lens was developed in and re-coded across several environments – early Silverlight with Javascript; later Silverlight with C#; and more recent work with HTML5 and Javascript – as well as its re-design based on user experience testing.

The original code used an image clip combined with ellipse geometry to create the lens shape and to use that same image clip area to show the past image. Note that this first iteration requires the lens to be selected with the mouse and is active only as long as the left mouse button is held down.

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Width="1280" Height="720"
        Background="White"
        x:Name="Page">
  <Canvas.RenderTransform>
    <TransformGroup>
      <ScaleTransform ScaleX="1.0" ScaleY="1.0" x:Name="PageScale"/>
    </TransformGroup>
  </Canvas.RenderTransform>
</Canvas>
```
The page’s Javascript, code-behind, is shown below. Note that some of these scaling options are to accommodate the facility to switch back and forth between full screen mode.

```javascript
if (!window.senseport_test_ideas)
    senseport_test_ideas = {};
senseport_test_ideas.Page.prototype = {
    handleLoad: function(control, userContext, rootElement)
    {
        this.control = control;

        // following settings are for enabling full screen setting
        m_root = rootElement.FindName("palimpsest");
        control.content.onFullScreenChange = FullScreenChange;
    }
}

// Global variables below used to keep track of the
// mouse position and whether the object is captured
// by the mouse.
var isMouseCaptured;
var mouseVerticalPosition;
var mouseHorizontalPosition;

// global variables below to ensure
// scaling works effectively
// including the (senseport/palimpsest) lens
var scaleX = 1;
var scaleY = 1;

function handleMouseDown (sender, args)
{
    var item = sender;
    mouseVerticalPosition = args.getPosition(null).y;
    mouseHorizontalPosition = args.getPosition(null).x;
    isMouseCaptured = true;
    item.CaptureMouse();
}

function handleMouseMove (sender, args)
{
    var item = sender;
    if (isMouseCaptured)
    {
        if (sender.Name == "senseport")
        {
            // Calculate the current position of the object.
            var deltaY = args.getPosition(null).y - mouseVerticalPosition;
            var deltaX = args.getPosition(null).x - mouseHorizontalPosition;
        }
    }
}
```
Later versions, which enhanced and improved the techniques, were coded in C# rather than JavaScript. These redeveloped programs also include modifications based on user experience testing and associated feedback. In particular, this code recasts the cursor as the lens and the lens is re-sizeable.

Later versions, which enhanced and improved the techniques, were coded in C# rather than JavaScript. These redeveloped programs also include modifications based on user experience testing and associated feedback. In particular, this code recasts the cursor as the lens and the lens is re-sizeable.

Later versions, which enhanced and improved the techniques, were coded in C# rather than JavaScript. These redeveloped programs also include modifications based on user experience testing and associated feedback. In particular, this code recasts the cursor as the lens and the lens is re-sizeable.

```xml
<UserControl
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="Senseport_NET_generic_version.MainPage">
    <Grid x:Name="LayoutRoot" Background="White" MouseMove="LayoutRoot_MouseMove"
        MouseWheel="LayoutRoot_MouseWheel" Cursor="None">
```

© 2013 Bath Spa University, Centre for Creative Computing
With the associated C# as follows:

```csharp
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;

namespace Senseport__NET_generic_version
{
    public partial class MainPage : UserControl
    {
        public MainPage()
        {
            // Required to initialize variables
            InitializeComponent();
        }

        private void LayoutRoot_MouseMove(object sender, MouseEventArgs e)
        {
            bigScene.Visibility = Visibility.Visible;
            double x = e.GetPosition(null).X;
            double y = e.GetPosition(null).Y;
            Ellipse.Center = new Point(x, y);
        }

        private void LayoutRoot_MouseWheel(object sender, MouseWheelEventArgs e)
        {
            if (e.Delta > 0)
            {
                Ellipse.RadiusX = Ellipse.RadiusX + 15;
                Ellipse.RadiusY = Ellipse.RadiusY + 15;
            }
            else
            {
                Ellipse.RadiusX = Ellipse.RadiusX - 15;
            }
        }
    }
}
```

Whilst the use of Silverlight provided a rapid prototyping environment, uncertainty about its future direction and durability resulted in a recoding of some of the techniques into an early HTML5 and Javascript based alternative. An illustration of this open standards approach, as applied to the lens technique, is listed below.
Videos on YouTube illustrate these programmatic techniques for HCI in use. The site fishenden.com also contains interactive examples.

**PHIDGETS PROTOTYPE**

Subsequent developments explored alternative ways for users to control and interact with techniques and content. For example, what if users were able to interact with content in non-physical ways – such as by moving around an installation space, or by making a noise or gesture – or by using alternative physical control surfaces, such as turning dials and knobs on a custom-designed piece of installation equipment? To explore these ideas in more detail, a prototyping environment that provided a range of sensors and controls was required, and Phidgets were chosen to meet this purpose. Phidgets provide and support a wide range of sensors, programmable from a variety of devices, including a PC.

Phidgets facilitate modular electronic building blocks for low cost USB sensing and control. They provide an inbuilt API that works across a diversity of programming languages, including C/C++, C#, Cocoa, Delphi, Flash AS3, Flex AS3, Java, LabVIEW, MATLAB, Max/MSP, MRS, Python, REALBasic, Visual Basic.NET, Visual Basic 6.0, Visual Basic for Applications, Visual Basic Script, and Visual C/C++/Borland.NET.

Initial programming utilised a mix of Javascript and C# programming languages, both of which can be used (alongside other languages) for developing Silverlight applications. However, Silverlight applications are designed to be cross-platform and the Silverlight environment is effectively sandboxed to achieve this: it does not support platform-specific APIs and hooks (since that would break cross-platform compatibility). Phidgets would not therefore work with

---


4 See [http://www.phidgets.com](http://www.phidgets.com)
Silverlight. However, Silverlight is the cross-platform derivative of the platform-specific Windows Presentation Foundation (or WPF). As the name suggests, WPF only runs on Windows, but is therefore able to incorporate the Phidgets device driver for Windows. For this reason, it was decided to utilise WPF in order to explore Phidgets as a potential method of providing an alternative HCI.

A 1018 Phidget Interface Kit 8/8/8 was acquired as the control centre for a range of Phidgets sensors. The 1018 connects directly to a computer via the USB port and provides 8 analogue inputs with configurable data acquisition rates, 8 digital inputs with hardware noise filtering, 8 digital outputs and a 5V terminal block beside analogue input 7 and digital output 7.

A range of sensors was also acquired (touch sensor, force sensor, slider sensor, precision light sensor, rotation sensor, magnetic sensor, sound sensor, sonar sensor) together with some generic switches and a mix of LEDs. Initial work concentrated on providing a physical interface to an existing compositional technique: the slider sensor was used with a composition example that previously utilised the mouse-controlled slider. Where the interaction had previously relied on movements of the mouse, this prototype hardware controller enabled the physical slider to be moved backwards and forwards, respectively revealing and hiding the hidden layers of a place over time.

With a variety of physical controls now available, the potential to provide a means of enabling additional interaction with visual layers was explored. Two sensors were used for this:

- the rotation sensor – mixing in a third layer (image) of the same place as it is rotated one way and removing it as the sensor is rotated in the other direction
- the force sensor – mixing in a fourth layer (image) of the same place when pressure is applied to the sensor and removing it when pressure ceases to be applied

In addition, the precision light sensor was programmed to randomly mix the two initial layers, adding a slightly unstable feel to the palimpsest navigation since changes in prevailing light conditions, or movements around the light sensor, resulted in fluctuations to the mix of layers. The touch sensor was used to provide a binary reveal/hide single-touch way of revealing the underlying visual image.

The idea of a physical timeline also emerged: for an installation, this might take the form of a line on the floor – the nearer the projection screen a user moves, the further back in time they travel; moving further away, the more they return to the present day. They would thus be able, dependent on their physical positioning, to “walk into the past”. This concept was explored with the Phidgets sonar sensor (able to detect objects from 0 cm to 6.45 meters with approximately 2.5 cm resolution).

The sonar sensor was prototyped with two alternative subroutines. In one, the closer to the sensor a person approached the further back in time the layer of images displayed. In the other, the reverse applied. Both models worked in initial tests, with some occasional fluctuation of the interplay of layers due to the lack of any smoothing function being applied to the sonar sensor (by design).

After the conclusion of these initial tests, a subset of the interface devices was subsequently fitted into basic hardware enclosures. The slider control and rotational control were fitted in the main enclosure, with the sonar sensor installed in its own dedicated enclosure so that it could be situated separately from the main control surfaces. Early testing using a
HD projection system and 86” display screen confirmed its suitability to provide a way of navigating the layers of time and place based on how far from a screen the user was situated. The sonar sensor calibrates on initiation and occasionally provided erratic readings, but this also created an interesting artistic effect, destabilising the layers as they blend and morph. For more precision control, it would be possible to use several sonar sensors configured in an array in such a way that they more accurately determine the physical location of a person and hence the way in which the installation interacts with them. A smoothing function could also be utilised to prevent the occasional fluctuation of readings.

![Figure 10: the Phidgets prototype HCI](image)

For the example where the sonar sensor was in slot 0 of the Phidgets USB controller, the following code extract illustrates the response mechanism and the adaptation of the respective blend of present and past images:

```csharp
if (test == 0)
{
    int SensorDistance = e.Value;
    Palimpsest_Opacity = 1 - ((float)SensorDistance / 100); //near=past, far=present
}
```

A video of the development work using Phidgets can be viewed on YouTube.

**PORTING TO KINECT**

Kinect is a motion and sound sensing input device suitable for programming with the Windows operating system environment. It is based on the Kinect sensor used with Xbox 360, but designed specifically for use with the Kinect SDK to enable programming of its features in a PC-based environment.

Webb and Ashley (2012) provide a range of useful C# sample code and templates for commencing Kinect user interaction programming, including (user) skeleton tracking and speech recognition. The Kinect palimpsest HCI prototyping work discussed in this paper was bootstrapped from their sample code, although updates to various deprecated method and function calls were required since the Kinect SDK utilised (version 1.7) was a later version than that on which their book is based.

For prototyping, the skeleton figure and hand were re-used from sample code in Webb and Ashley to provide on-screen visualisation confirmation of the way user interaction was controlling both the slider and the lens techniques as they

---

1 See [http://youtu.be/m9f3_H6ZY8](http://youtu.be/m9f3_H6ZY8)
were ported to the Kinect environment. Later, once the techniques had been improved and refined, the on-screen skeleton would be removed. The presence of the on-screen hand was subject to some further iterative investigation since its presence on the screen might enable users to better understand how their gestures were related to the on-screen reactions of the slider and the lens. However, its opacity was reduced to make it less dominant and to enable images to be seen behind the hand rather than obscuring them.

Code was re-used from the earlier WPF/Phidgets HCI programming and development work. The initial focus was on the slider, with user left/right hand movements used to mimic the effect of the former physical slider control, enabling an older image to emerge and dominate the screen with leftwards gesture movements and to fade and disappear with rightwards gesture movements.

The core of the Kinect enabled version of the slider code is shown in the listing below.

```csharp

DepthImagePoint point =
this._KinectDevice.CoordinateMapper.MapSkeletonPointToDepthPoint(hand.Position,
this._KinectDevice.DepthStream.Format);
point.X = (int)((point.X * LayoutRoot.ActualWidth /_KinectDevice.DepthStream.FrameWidth) - (HandCursorElement.ActualWidth / 2.0));
point.Y = (int)((point.Y * LayoutRoot.ActualHeight / _KinectDevice.DepthStream.FrameHeight) - (HandCursorElement.ActualHeight / 2.0));

Canvas.SetLeft(HandCursorElement, point.X);
Canvas.SetTop(HandCursorElement, point.Y);

txtStatus.Text = "X is: " + point.X;
image1.Opacity = point.X / LayoutRoot.ActualWidth;
```

![Figure 11: Kinect palimpsest slider prototyping](image)

A video of the prototyping of the Kinect slider can be viewed on YouTube⁶.

Once the slider prototype was successfully operational, attention turned to porting of the lens technique. As no WPF code existed for the lens (since it had not previously been prototyped with the Phidget HCI), appropriate code was

instead redeveloped from a combination of the Silverlight code for the lens and the WPF code for the slider. Gesture movements of the user’s hand nearest the screen moves the lens around, the area over which it passes providing the same ‘lens through time’ as earlier experimentation – presenting an underlying layer comprising an earlier image of the same scene.

The construction of the layout of the images and the lens are shown in the markup code below.

```xml
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:c="clr-namespace:VoeTek.Kinect.PalimpsestLens"
  mc:Ignorable="d">
  <Window.Title>Kinect Palimpsests Alpha 1 - Fishenden</Window.Title>
  <Window.Background>White</Window.Background>
  <Viewbox>
    <Grid x:Name="LayoutRoot" Width="1920" Height="1200" Background="White" HorizontalAlignment="Center" VerticalAlignment="Center" />
    <Image x:Name="image2" Stretch="Fill" Source="Images/chiswick2.jpg" Opacity="1" HorizontalAlignment="Center" VerticalAlignment="Center" />
    <Grid x:Name="bigScene" Opacity="0.9" Visibility="Collapsed">
      <Image x:Name="image1" Stretch="Fill" Source="Images/chiswick1.jpg">
        <Image.Clip>
          <EllipseGeometry x:Name="Ellipse" RadiusX="215" RadiusY="215" Center="300,350" />
        </Image.Clip>
      </Image>
    </Grid>
    <Canvas x:Name="magGlass" Width="260" Height="395" Canvas.Left="-1" Canvas.Top="-1" />
    <Path x:Name="magnifyArea" Fill="#00000000" Stretch="Fill" Canvas.Left="-1" Canvas.Top="-1" Data="M1.5202086, -0.38218391 L -0.47718381, 602.46073 302.12777, 600.46456 324.01043, 527.0249 476.98435, 527.0249 499.86962, 599.46648 800.47717, 601.46264 800.47718, -0.38218391 Z" />
    <c:SkeletonViewer x:Name="SkeletonViewerElement" />
    <Polyline x:Name="CrayonElement" Stroke="Black" StrokeThickness="15" />
    <Canvas x:Name="GameBoardElement">
      <Canvas x:Name="PuzzleBoardElement" />
      <Image x:Name="HandCursorElement" Source="Images/hand.png" RenderTransformOrigin="0.5,0.5" Canvas.Left="134.694" Canvas.Top="355.102">
        <Image.RenderTransform>
          <TransformGroup>
            <ScaleTransform x:Name="HandCursorScale" ScaleX="1" ScaleY="1" />
          </TransformGroup>
        </Image.RenderTransform>
      </Image>
    </Canvas>
    <TextBox x:Name="txtStatus" HorizontalAlignment="Left" Height="52" Margin="228,-75,0,0" TextWrapping="Wrap" Text="test&#x26;#xA;" VerticalAlignment="Top" Width="190" FontSize="24" FontWeight="Bold" />
    <GameBoardElement />
    <CrayonElement />
    <HandCursorElement />
    <SkeletonViewerElement />
  </Viewbox>
</Window>
```

The core of the resulting Kinect enabled version of the lens programmatic code is shown in the listing below.

```csharp

DepthImagePoint point =
  this._KinectDevice.CoordinateMapper.MapSkeletonPointToDepthPoint(hand.Position, this._KinectDevice.DepthStream.Format);
point.X = (int)((point.X * LayoutRoot.ActualWidth / _KinectDevice.DepthStream.FrameWidth) - (HandCursorElement.ActualWidth / 2.0));
point.Y = (int)((point.Y * LayoutRoot.ActualHeight / _KinectDevice.DepthStream.FrameHeight) - (HandCursorElement.ActualHeight / 2.0));
Canvas.SetLeft(HandCursorElement, point.X);
```
Canvas.SetTop(HandCursorElement, point.Y);

Canvas.SetTop(HandCursorElement, point.Y);

txtStatus.Text = "X is: " + point.X;

bigScene.Visibility = Visibility.Visible;

Ellipse.Center = new Point(point.X, point.Y);

Figure 12: Kinect palimpsest lens prototyping

A video of the prototyping of the Kinect lens can be viewed on YouTube7.

REFINING WITH Kinect

Earlier HCI work experimented with differing techniques for switching between navigation techniques – for example, the work with Phidgets used different physical controllers to enable different methods of interaction (sonar sensors for manipulation based on user distance from the screen; a physical slider to mimic the earlier on-screen slider control; knobs for merging different layers, etc.). With Kinect’s support for speech recognition, a new method of enabling users to switch between the two techniques – lens and slider – became possible, with the option of enabling users, for example, merely to say “lens” or “slider” in order to switch their mode of interaction.

To support this, the previously separate code for the lens and the slider were merged into a single program. To validate this code, initially an onscreen button was used to switch between lens or slider modes of operation. Having successfully tested the code, the button was removed and in its place work started on utilising Kinect’s speech recognition features.

Kinect Speech recognition

Kinect’s speech recognition features rely upon the Microsoft.Speech library built around the SpeechRecognitionEngine. Speech recognition relies on the use of ‘grammars’, which can be either single words or a string of words. It also supports the use of wildcards – so ‘slider*’ would enable the recognition of both ‘slider’ or ‘sliders’. There is also a Choices class that is used to specify alternative multiple values that can be added to the grammar vocabulary for speech recognition. So for example:

7 See http://youtu.be/iRd.gzkP3JH
var choices = new Choices;
choices.add("lens");
choices.add("slider");

var grammarBuilder = new GrammarBuilder();
grammarBuilder.Append(choices);

var grammar = new Grammar(grammarBuilder);

The engine goes through several stages of recognition: SpeechHypothesized, SpeechRecognized or SpeechRecognitionRejected. Essentially these operate as ‘gates’, testing what the engine thinks it has heard and then either trying to match it against the grammar or rejecting it as not sufficiently close to offer a match. A degree of confidence is available and for initial testing the confidence levels were displayed on screen to check how well the speech recognition engine was matching the spoken words “lens” and “slider” and hence reliably switching between these modes.

Alongside this work, the on-screen skeletal figure was removed since this was not part of the intended user experience and merely utilised during development to assist with improving the performance of the code. As well as enabling the user to switch between the lens and slider technique by speaking aloud ‘use slider’ or ‘use lens’, additional past and present images from another location were also added. To enable the user to switch the location as well as the technique, they can say ‘location London’ or ‘location Leicester’ and the images change to the desired location, inheriting the navigational technique currently in use.

private void CreateGrammars(RecognizerInfo ri)
{
    //user oral instruction format is {"use lens/slider"}, {"location London/Leicester"}
    var technique = new Choices();
    technique.Add("lens");
    technique.Add("slide");
    technique.Add("London");
    technique.Add("Leicester");

    var create = new Choices();
    create.Add("location");
    create.Add("use");
    var gb = new GrammarBuilder();
    gb.Culture = ri.Culture;
    gb.Append(create);
    gb.AppendWildcard();
    gb.Append(technique);
    var g = new Grammar(gb);
    _sre.LoadGrammar(g);
}

CURRENT AND FUTURE WORK

Further options to enhance the user experience of the Kinect HCI environment are currently in development. These include testing additional interaction techniques, such as tracking the distance of a user from the sensor and growing or shrinking the size of the lens depending on whether someone's hand moves closer to or further from the screen. Based on earlier work, which demonstrated the useful impact of associated audio in improving the experience of palimpsests navigation techniques (Fishenden and Hugill), work is also progressing to add audio, including audio that will come to the foreground as the lens passes over particular visual items (for example, emphasising the sound of children playing as the lens passes over children in the images, or emphasising the sound of horses hooves when passing over a horse and cart in the earlier underlying image).
BIBLIOGRAPHY

Fishenden, J and Hugill, A. Palimpsests of Time and Place. 2011. IEEE COMPSAC

ACKNOWLEDGEMENTS

Earlier work described in this paper took place as part of doctoral research at the Institute of Creative Technologies, De Montfort University, during 2008-2012. Work with Kinect is a component of recent and ongoing research in the Centre for Creative Computing Collaboratory at Bath Spa University.

ABOUT THE CODE

The computer programming code used in this paper is work-in-progress and not intended as production code. It is provided “as is” and without any warranties or guarantees, either express or implied, about its fitness for purpose. It may be freely re-used for non-commercial purposes if desired, provided that full credit is given regarding its sources.

ABOUT THE AUTHOR

Dr Jerry Fishenden CITP FBCS FIAP FRSA is a Senior Research Fellow at the Centre for Creative Computing, Bath Spa University.