



Kinect for Windows

305890
Spring 2014
4/01/2014
Kyoung Shin Park

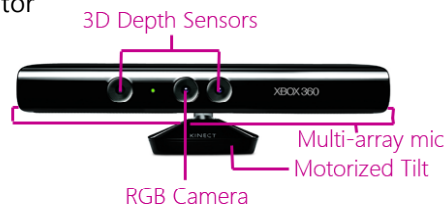
What is the Kinect?

- Motion sensing device for XBOX360 and Windows
- Natural User Interface for interacting with the XBOX360 and Windows PC by using gestures and body movement, instead of a controller



The Kinect Sensor

- The Kinect sensor device consists of
 - Color (RGB) camera (30fps@640x480 or 15fps@1280x1024),
 - Depth sensor (infrared projector & camera)
 - Multi-array microphone.
 - Motorized tilt sensor – play space control is done through a tilt motor



The Kinect Sensor



Kinect Video Output

- 30 Hz frame rate; 57 degree FOV (field of view)



8-bit VGA RGB (640x480)



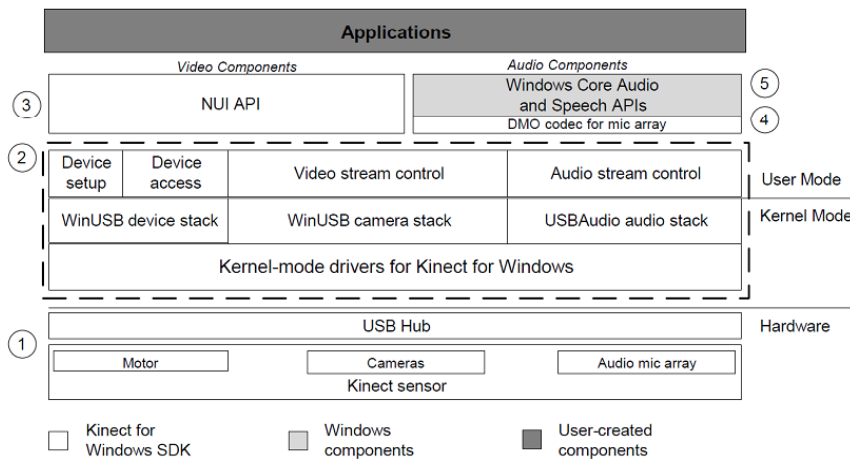
12-bit Monochrome (320x240)

Kinect Audio System

- 4 microphone array with hardware-based audio process
 - Multichannel echo cancellation
 - Sound position tracking
 - Other digital signal processing (noise suppression and reduction)



Kinect SDK Architecture

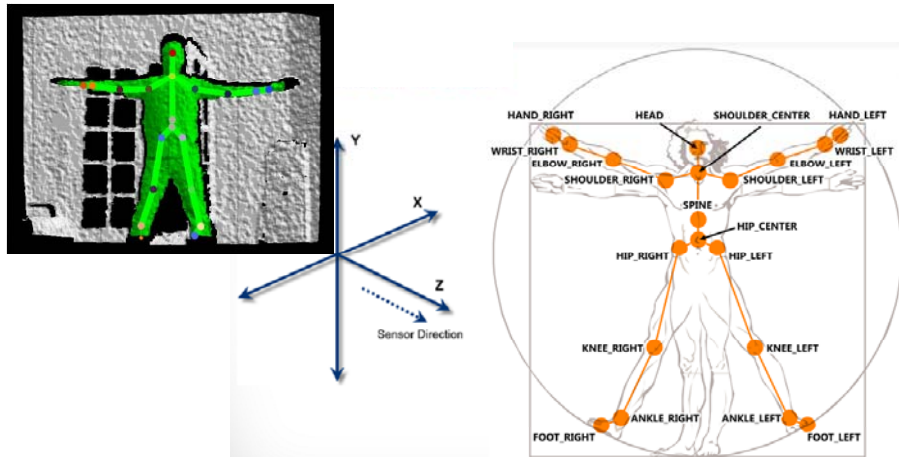


Kinect SDK

- Kinect SDK provides data streams:
 - RGB stream at 640x 480 resolution (32 bits per pixel)
 - Depth stream at 320x240 resolution (16 bits per pixel – depth represented in 12 bits)
 - Skeletal tracking capabilities (e.g., skeleton joint positions)
 - Echo-canceled audio beam data
 - Speech recognition
 - And others (frame #s, timestamps, tilt sensor data)

Skeleton Tracking

- Kinect SDK provides

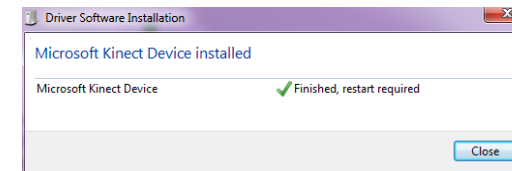


Connecting the Kinect

- Get a Kinect for Windows Sensor
- Download and install the Kinect for Windows SDK (version 1.7, March 18, 2013)
 - <http://www.microsoft.com/en-us/kinectforwindows/develop/developer-downloads.aspx>

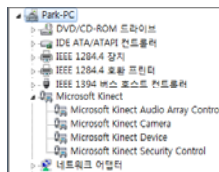


- Connect the Kinect device using the adapter for USB and Automatic Update will install the drivers

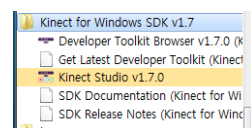


Connecting the Kinect

- If correctly installed, you can see the device in Device Manager

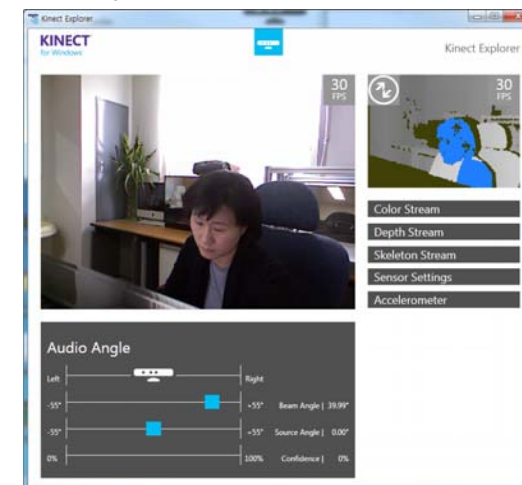


- Now start the Kinect Sample Browser, select C# and run some of the examples to make sure it's correctly installed



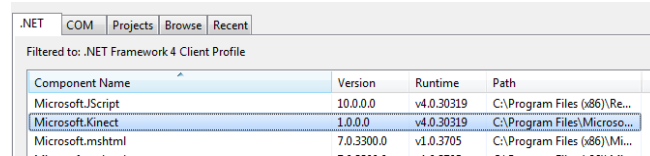
Connecting the Kinect

- Run Kinect Explorer-WPF



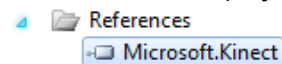
Your XNA4.0 Program using Kinect

- Create a new XNA window game
 - File->New->Projects
 - Visual C# -> XNA Game Studio 4.0 -> Windows Game (4.0)
 - Give it a project name (e.g. "KinectBasic")
- To use the Kinect SDK, you will need to add a reference to it



Component Name	Version	Runtime	Path
Microsoft.JScript	10.0.0.0	v4.0.30319	C:\Program Files (x86)\Re...
Microsoft.Kinect	1.0.0.0	v4.0.30319	C:\Program Files\Microso...
Microsoft.mshtml	7.0.3300.0	v1.0.3705	C:\Program Files (x86)\Mi...

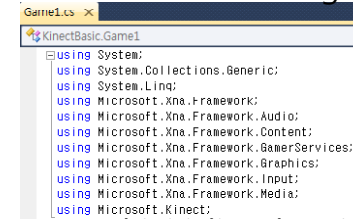
- You will see it in the project reference



13

Your XNA4.0 Program using Kinect

- You need to add a "using statement"



- Create and initialize the KinectSensor

```
protected override void Initialize()
{
    // TODO: Add your initialization logic here
    KinectSensor.KinectSensors.StatusChanged += new EventHandler<StatusChangedEventArgs>(KinectSensors_StatusChanged);
    DiscoverKinectSensor();

    base.Initialize();
}

void KinectSensors_StatusChanged(object sender, StatusChangedEventArgs e)
{
    if (this.kinectSensor == e.Sensor)
    {
        if (e.Status == KinectStatus.Disconnected ||
            e.Status == KinectStatus.NotPowered)
        {
            this.kinectSensor = null;
            this.DiscoverKinectSensor();
        }
    }
}
```

Your XNA4.0 Program using Kinect

- Also create a function called DiscoverKinectSensor()

```
private void DiscoverKinectSensor()
{
    foreach (KinectSensor sensor in KinectSensor.KinectSensors)
    {
        if (sensor.Status == KinectStatus.Connected)
        {
            // Found one, set our sensor to this
            kinectSensor = sensor;
            break;
        }
    }

    if (this.kinectSensor == null)
    {
        connectedStatus = "Found none Kinect Sensors connected to USB";
        return;
    }

    // You can use the kinectSensor.Status to check for status
    // and give the user some kind of feedback
    switch (kinectSensor.Status)
    {
        case KinectStatus.Connected:
        {
            connectedStatus = "Status: Connected";
            break;
        }
        case KinectStatus.Disconnected:
        {
            connectedStatus = "Status: Disconnected";
            break;
        }
        case KinectStatus.NotPowered:
        {
            connectedStatus = "Status: Connect the power";
            break;
        }
        default:
        {
            connectedStatus = "Status: Error";
            break;
        }
    }

    // Init the found and connected device
    if (kinectSensor.Status == KinectStatus.Connected)
    {
        InitializeKinect();
    }
}
```

Your XNA4.0 Program using Kinect

- Also create DiscoverKinectSensor, InitializeKinect

```
private void DiscoverKinectSensor()
{
    foreach (KinectSensor sensor in KinectSensor.KinectSensors)
    {
        if (sensor.Status == KinectStatus.Connected)
        {
            // Found one, set our sensor to this
            kinectSensor = sensor;
            break;
        }
    }

    if (this.kinectSensor == null)
    {
        connectedStatus = "Found none Kinect Sensors connected to USB";
        return;
    }

    // You can use the kinectSensor.Status to check for status
    // and give the user some kind of feedback
    switch (kinectSensor.Status)
    {
        case KinectStatus.Connected:
        {
            connectedStatus = "Status: Connected";
            break;
        }
        case KinectStatus.Disconnected:
        {
            connectedStatus = "Status: Disconnected";
            break;
        }
        case KinectStatus.NotPowered:
        {
            connectedStatus = "Status: Connect the power";
            break;
        }
        default:
        {
            connectedStatus = "Status: Error";
            break;
        }
    }

    // Init the found and connected device
    if (kinectSensor.Status == KinectStatus.Connected)
    {
        InitializeKinect();
    }
}
```

Your XNA4.0 Program using Kinect

■ Get images from the Kinect RGB camera

```
void KinectSensor.ColorFrameReady(object sender, ColorImageFrameReadyEventArgs e)
{
    using (ColorImageFrame colorImageFrame = e.OpenColorImageFrame())
    {
        if (colorImageFrame != null)
        {
            byte[] pixelsFromFrame = new byte[colorImageFrame.PixelDataLength];
            colorImageFrame.CopyPixelDataTo(pixelsFromFrame);

            Color[] color = new Color[colorImageFrame.Height * colorImageFrame.Width];
            KinectRGBVideo = new Texture2D(GraphicsDevice, colorImageFrame.Width, colorImageFrame.Height);

            // Go through each pixel and set the bytes correctly
            // Remember, each pixel got a Red, Green and Blue
            int index = 0;
            for (int y = 0; y < colorImageFrame.Height; y++)
            {
                for (int x = 0; x < colorImageFrame.Width; x++, index += 4)
                {
                    color[y * colorImageFrame.Width + x] = new Color(pixelsFromFrame[index + 2], pixelsFromFrame[index + 1], pixelsFromFrame[index + 0]);
                }
            }

            // Set pixeldata from the ColorImageFrame to a Texture2D
            KinectRGBVideo.SetData(color);
        }
    }
}
```

Your XNA4.0 Program using Kinect

■ Create Texture2D & SpriteFont object in LoadContent()

```
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game content here
    KinectRGBVideo = new Texture2D(GraphicsDevice, 1337, 1337);
    font = Content.Load<SpriteFont>("SpriteFont1");
}
```

■ Stop & Dispose KinectSensor in UnloadContent()

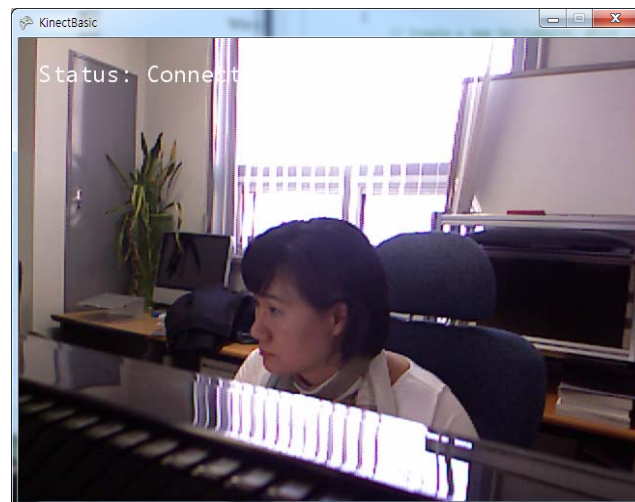
```
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
    KinectSensor.Stop();
    KinectSensor.Dispose();
}
```

■ Draw RGB video image & status text

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here
    spriteBatch.Begin();
    spriteBatch.Draw(KinectRGBVideo, new Rectangle(0, 0, 640, 480), Color.White);
    spriteBatch.DrawString(font, connectedStatus, new Vector2(20, 20), Color.White);
    spriteBatch.End();
    base.Draw(gameTime);
}
```

Your XNA4.0 Program using Kinect



XNA4.0 Kinect Skeleton Tracking

■ Start it from the previous "KinectBasic" project

■ Enable the SkeletonStream on our KinectSensor in InitializeKinect()

```
private bool InitializeKinect()
{
    // Color stream
    KinectSensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
    KinectSensor.ColorFrameReady += new EventHandler<ColorImageFrameReadyEventArgs>(KinectSensor_ColorFrameReady);

    // Skeleton Stream
    KinectSensor.SkeletonStream.Enable(new TransformSmoothParameters()
    {
        Smoothing = 0.5f,
        Correction = 0.5f,
        Prediction = 0.5f,
        JitterRadius = 0.05f,
        MaxDeviationRadius = 0.04f
    });
    KinectSensor.SkeletonFrameReady += new EventHandler<SkeletonFrameReadyEventArgs>(KinectSensor_SkeletonFrameReady);

    try
    {
        KinectSensor.Start();
    }
    catch
    {
        connectedStatus = "Unable to start the Kinect Sensor";
        return false;
    }
    return true;
}
```

XNA4.0 Kinect Skeleton Tracking

- Track skeleton joints
 - Kinect returns a position between -1 (left) and 1 (right)

```
void KinectSensor_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
{
    using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())
    {
        if (skeletonFrame != null)
        {
            //int skeletonSlot = 0;
            Skeleton[] skeletonData = new Skeleton[skeletonFrame.SkeletonArrayLength];

            skeletonFrame.CopySkeletonDataTo(skeletonData);
            Skeleton playerSkeleton = (from s in skeletonData where s.TrackingState == SkeletonTrackingState.Tracked select s).FirstOrDefault();
            if (playerSkeleton != null)
            {
                Joint rightHand = playerSkeleton.Joints[JointType.HandRight];
                righthandPosition = new Vector2(((0.5f * rightHand.Position.X) + 0.5f) * (640), (((-0.5f * rightHand.Position.Y) + 0.5f) * (480)));
                Joint leftHand = playerSkeleton.Joints[JointType.HandLeft];
                lefthandPosition = new Vector2(((0.5f * leftHand.Position.X) + 0.5f) * (640), (((-0.5f * leftHand.Position.Y) + 0.5f) * (480)));
            }
        }
    }
}
```

- Create Texture2D for hands

```
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game content here

    KinectRGBVideo = new Texture2D(GraphicsDevice, 1337, 1337);
    font = Content.Load("SpriteFont1");
    righthand = Content.Load<Texture2D>("right_hand");
    lefthand = Content.Load<Texture2D>("left_hand2");
}
```

21

XNA4.0 Kinect Skeleton Tracking

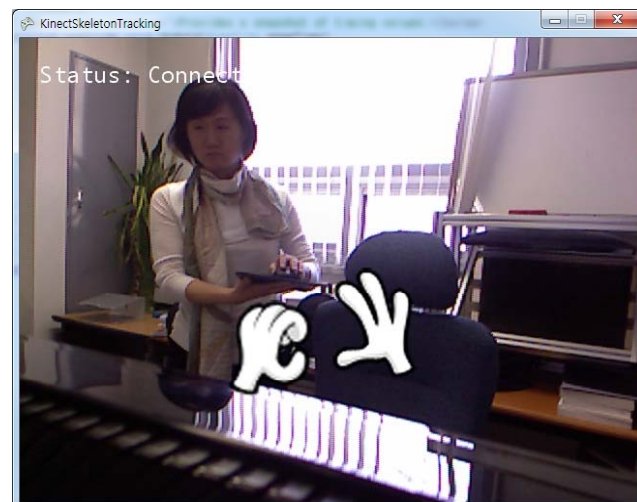
- Draw RGB video image & right/left hand textures

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here
    spriteBatch.Begin();
    spriteBatch.Draw(KinectRGBVideo, new Rectangle(0, 0, 640, 480), Color.White);
    spriteBatch.DrawString(font, connectedStatus, new Vector2(20, 20), Color.White);
    spriteBatch.Draw(righthand, righthandPosition, Color.White);
    spriteBatch.Draw(lefthand, lefthandPosition, Color.White);
    spriteBatch.End();

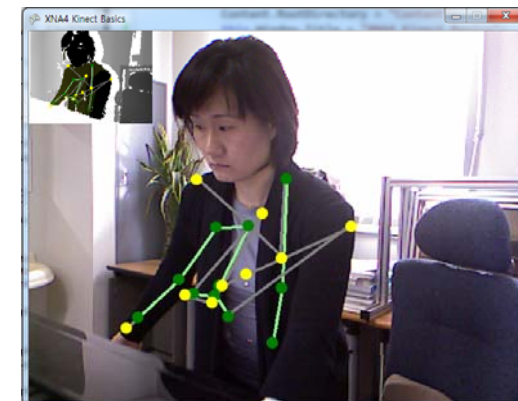
    base.Draw(gameTime);
}
```

XNA4.0 Kinect Skeleton Tracking



XNABasics

- Demonstrates the Kinect's ColorImageStream, DepthImageStream, SkeletonStream
<http://msdn.microsoft.com/en-us/library/jj131040>



InputHandler

```
protected void UpdateInput()
{
    float speed = 50.0f;
    if (inputs.PauseGame)
    {
        this.Exit();
    }
    if (inputs.IsKeyPressed(Keys.Up))
    {
        modelPosition += Vector3.Up * speed;
        System.Diagnostics.Trace.WriteLine("modelPosition=" + modelPosition);
    }
    else if (inputs.IsKeyPressed(Keys.Down))
    {
        modelPosition += Vector3.Down * speed;
        System.Diagnostics.Trace.WriteLine("modelPosition=" + modelPosition);
    }
    else if (inputs.IsKeyPressed(Keys.Right))
    {
        modelPosition += Vector3.Right * speed;
        System.Diagnostics.Trace.WriteLine("modelPosition=" + modelPosition);
    }
    else if (inputs.IsKeyPressed(Keys.Left))
    {
        modelPosition += Vector3.Left * speed;
        System.Diagnostics.Trace.WriteLine("modelPosition=" + modelPosition);
    }
}

protected override void Initialize()
{
    /// <summary>
    /// inputs component
    /// </summary>
    inputs = new InputHandler(this);
    Components.Add(inputs);
    /// <summary>
    /// fps component
    /// </summary>
    fpsCounter = new FpsCounter(this);
    Components.Add(fpsCounter);
    base.Initialize();
}
```

KinectHandler

```
protected override void Initialize()
{
    /// <summary>
    /// kinect add component
    /// </summary>
    kinect = new KinectHandler(this);
    kinect.ScreenCenter = new Vector2(graphics.PreferredBackBufferWidth / 2, graphics.PreferredBackBufferHeight / 2); // screen center
    Components.Add(kinect);
    /// <summary>
    /// inputs add component
    /// </summary>
    inputs = new InputHandler(this);
    Components.Add(inputs);
    /// <summary>
    /// fps add component
    /// </summary>
    fpsCounter = new FpsCounter(this);
    Components.Add(fpsCounter);
    base.Initialize();
}

// kinect
modelPosition.X -= kinect.LeftHandMoved.X * 5000.0f;
modelPosition.Y -= kinect.LeftHandMoved.Y * 5000.0f;
modelRotation -= kinect.RightHandMoved.X;
modelRotationPitch -= kinect.RightHandMoved.Y;
```

Reference

- <http://digitalerr0r.wordpress.com/2011/06/20/kinect-fundamentals-1-installation-setup/>
- <http://digitalerr0r.wordpress.com/2011/06/20/kinect-fundamentals-2-basic-programming/>
- <http://digitalerr0r.wordpress.com/2011/12/13/kinect-fundamentals-4-implementing-skeletal-tracking/>