Computer Science Capstone 2013

KINECTS IN UNITY

Design Specifications

Team Hex Pistols
Anthony Narsi
Jerry Boyang Peng
Sea Pong
Alexander Scarlett
Kevin Sheridan

Last Revision: March 7, 2013
Based on github code branch: `jerrypeng/capstone/with_calibration_code`
and `kdsheridan/unitycapstone/master`

Table of Contents

1 Introduction

1.1 Product Overview

This project will create an integrated system of multiple Microsoft Kinect sensors and large format displays to create a naturalized viewing of 3D panoramas on non-panoramic screens, similar to looking outside a 'virtual' window. NASA's astrophysicists are looking forward to the day where they can simply enter a room, flip a switch, and be surrounded by a previously unreachable territory like Mars.

Kinects in Unity is setting the basis for these scientists to be virtually immersed in a complex augmented reality, just as any other scientist could simply go to a lab or field and perform experiments. The main intention of this project is to allow a user to realistically experience an environment that he or she could not otherwise be physically present in. The Human-Computer Interaction Department at JPL wanted to simplify the process of tracking movements by cameras and transferring these movements to an augmented reality. Simplifying the process entails decreasing the cost of the system as a whole. NASA is working with Microsoft Kinects to prove to astrophysicists that entering a virtual environment does not require highly expensive camera and tracking equipment. Our project provides the foundation for this significant benefit.

1.2 Definitions, Acronyms, Abbreviations

See *Appendix A. Glossary*

1.3 References

Kinect SDK Documentation
(**http://www.microsoft.com/en-us/kinectforwindows/develop/resources.aspx**)

Unity 3D API Documentation
(**http://unity3d.com/company/support/documentation/**)

1.4 Document Overview

This document will detail the specifics of the Kinects in Unity application. It will include descriptions of each of the functions used in Kinect SDK and in Unity3d. The function descriptions will also cover the hierarchy of the code and the flow of data in our application. The document also details how the user interacts with the application.

The remainder of the document provides detailed specifications regarding the design of the Kinects in Unity application. The document includes descriptions of all project modules and explanations for all interactions between modules. The document will specify the user of each module and the module will be interacted with by such a user.

1.5 Team Contact Information

Chandra Krintz (Professor)
ckrintz@gmail.com

Stratos Dimopoulos (Teaching Assistant)
stratos.dimopoulos@gmail.com

Victor Luo (Mentor)
victor.luo@jpl.nasa.gov

Jerry Peng
jerry.boyang.peng@gmail.com

Sea Pong
sea@seapong.com

Alex Scarlett
scarlett.alex@gmail.com

Anthony Narsi
ajnarsi01@gmail.com

Kevin Sheridan
kdsheridan37@gmail.com

2 Design Overview

2.1 Introduction

The execution of Kinects in Unity can be summed up in two steps. The first is to process raw data from multiple Microsoft Kinects to track one user's location. The second is to show the user's observation windows of a 3D scene provided by multiple displays.

The project will be developed in Visual Studio 2012 Ultimate and Unity 3D Development Studio. The code will consist of mainly C# with some C++ used in low-level wrapper functions. The usage of C# means this project is *heavily* object-oriented. For an overview of what object-oriented programming entails and its general code structure, please visit: http://en.wikipedia.org/wiki/Object-oriented_programming. A decent knowledge and understanding of object-oriented is required for the comprehension of this design specification document.

Stretch goals include increasing the number of Kinects for a wider range of view and increasing the number of users present in the workspace.

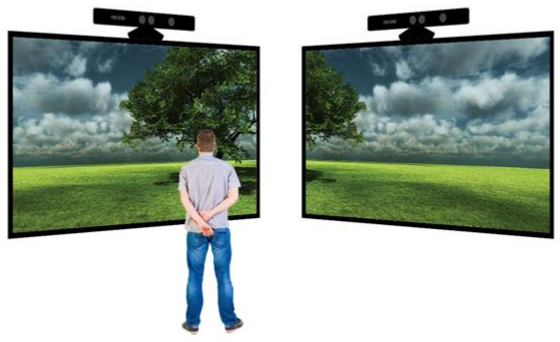


Figure 2. User Standing on Left Looking Right

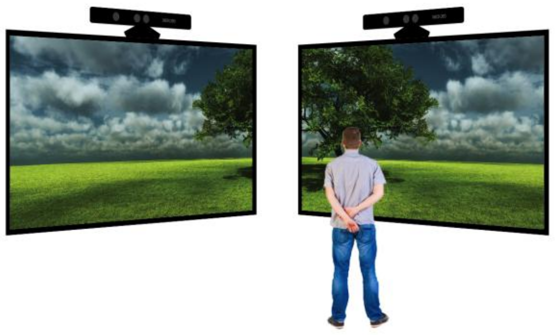
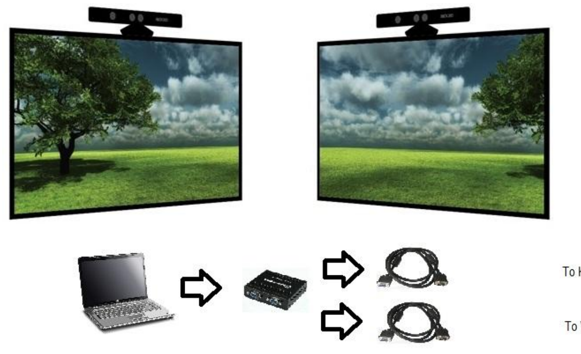


Figure 1. User Standing on Right Looking Left

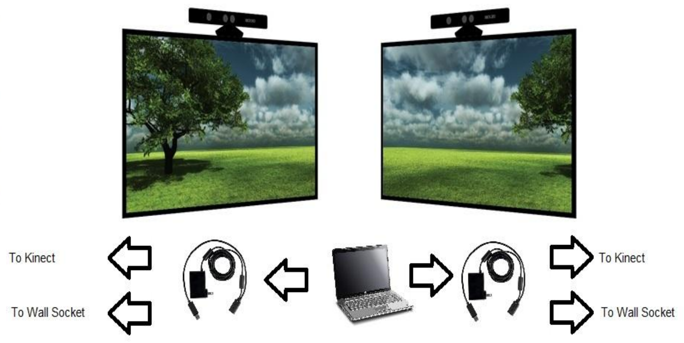
As seen in the figures above, the user has no input to the system, with the exception of his movement. The controls of the system are completely invisible to the user, and results are immediately effective.

2.2 Hardware Diagram

The hardware setup is very simple. To use a dual monitor setup with one computer, an additional device is needed to split the computer's VGA output into 2 separate VGA outputs. The device recommended to use is called the DualHead2Go. Simply connect VGA cables to the TVs and attached them to the DualHead2Go. Then attach a VGA cable from the computer to DualHead2Go. The Kinects should also attach to the same computer through USB. If the computer does not have 2 free USB ports, a USB hub should be used. It is recommended that the Kinects be directly attached to the computer while other USB devices be attached to the USB hub.



VGA-Laptop-Display Connection



USB-Kinect-Power Connection

2.3 Implementation Strategy

As previously mentioned, this project will be broken up into two major parts. The Kinect data processing module and the Unity front end rendering module. The two modules will be joined together via a wrapper class.

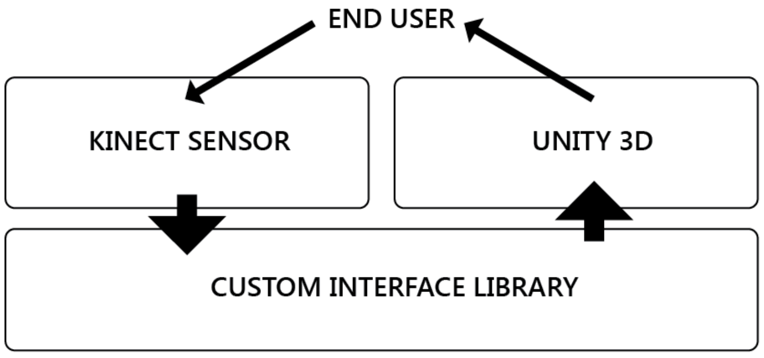


Figure 3. Implementation Strategy

2.4 Data Flow Diagram

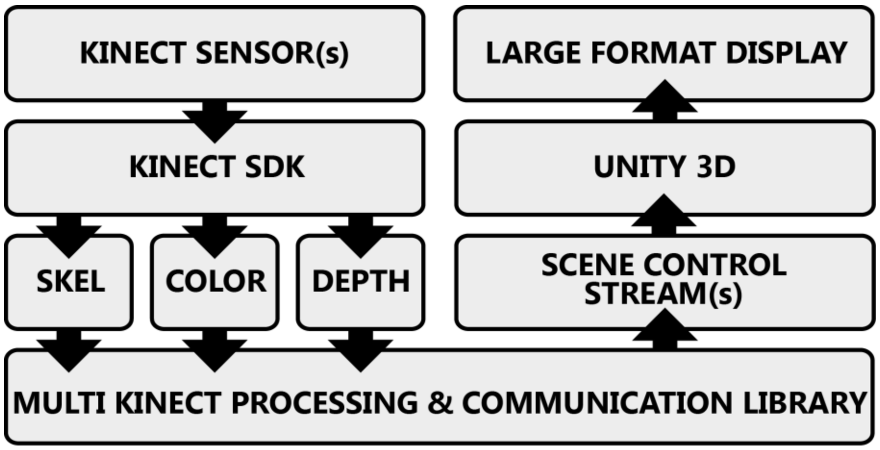


Figure 4. Software Data Flow Diagram

2.5 Global Class Diagram (UML)

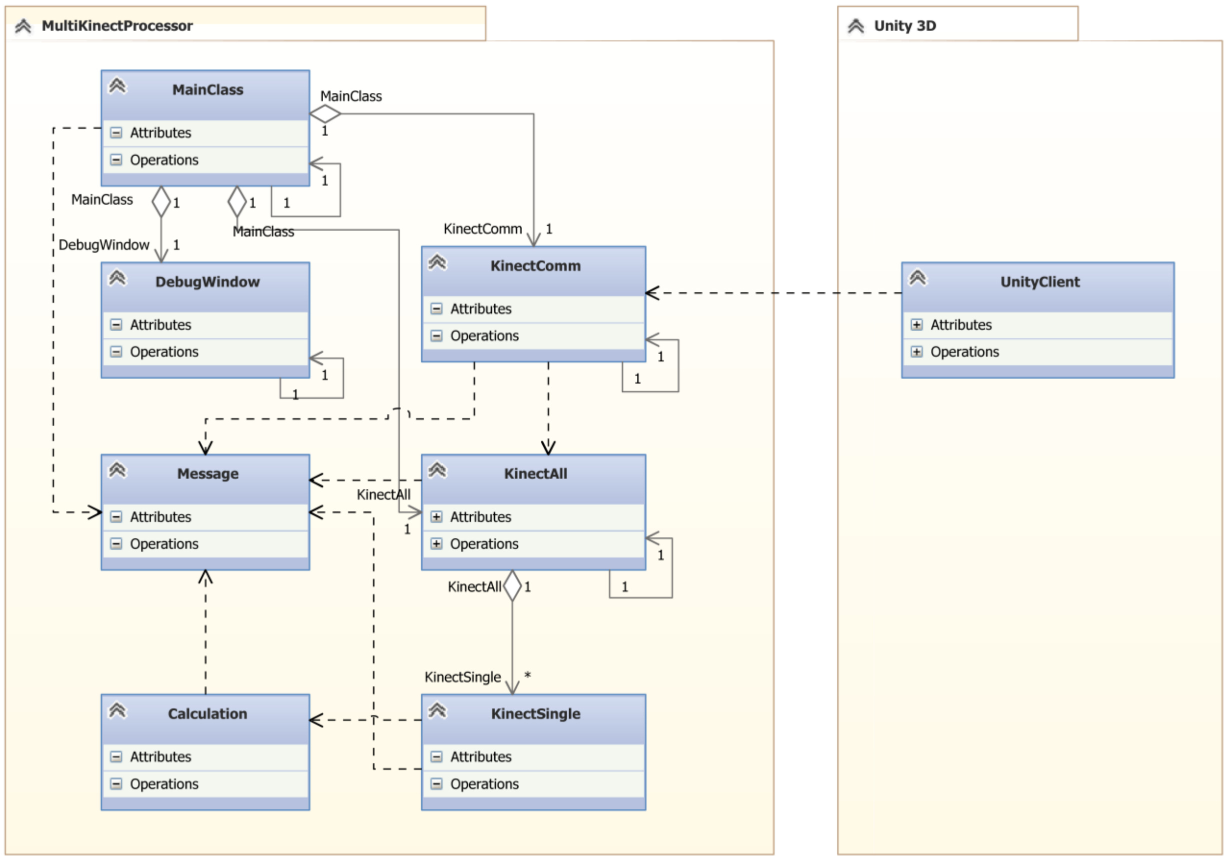


Figure 5. Global Class Diagram (UML)

2.6 Application Execution State Diagram

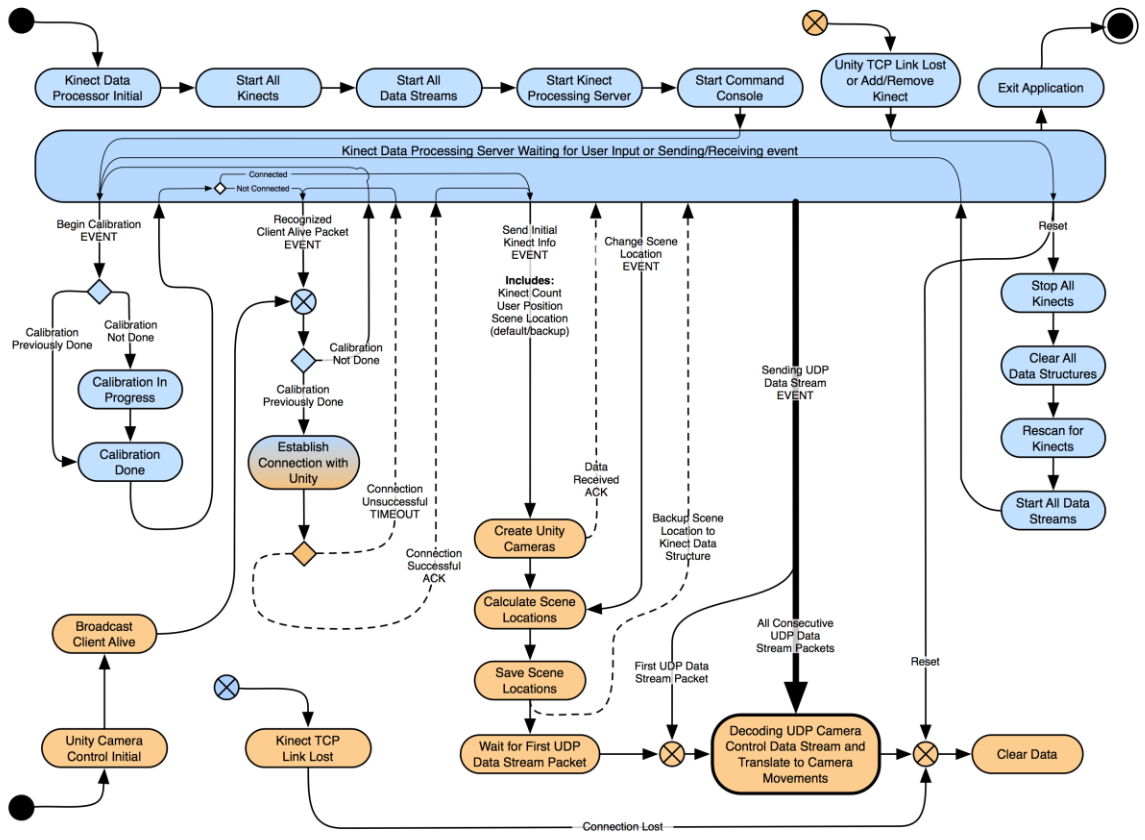

Kinect Data Processor Initial
Start All Kinects
Start All Data Streams
Start Kinect Processing Server
Start Command Console
Unity TCP Link Lost or Add/Remove Kinect
Exit Application
Kinect Data Processing Server Waiting for User Input or Sending/Receiving event
Connected
Not Connected
Begin Calibration EVENT
Calibration Previously Done
Calibration Not Done
Calibration In Progress
Calibration Done
Recognized Client Alive Packet EVENT
Calibration Not Done
Calibration Previously Done
Establish Connection with Unity
Connection Unsuccessful TIMEOUT
Connection Successful ACK
Broadcast Client Alive
Unity Camera Control Initial
Kinect TCP Link Lost
Send Initial Kinect Info EVENT
Includes:
Kinect Count
User Position
Scene Location (default/backup)
Change Scene Location EVENT
Data Received ACK
Create Unity Cameras
Calculate Scene Locations
Save Scene Locations
Wait for First UDP Data Stream Packet
Backup Scene Location to Kinect Data Structure
Sending UDP Data Stream EVENT
First UDP Data Stream Packet
All Consecutive UDP Data Stream Packets
Decoding UDP Camera Control Data Stream and Translate to Camera Movements
Reset
Stop All Kinects
Clear All Data Structures
Rescan for Kinects
Start All Data Streams
Reset
Clear Data
Connection Lost

Figure 6. Application Execution State Diagram

3 Design Specification

3.1 Kinect SDK Core Code

3.1.1 Overview

The Microsoft Kinect SDK is used to receive three streams of data (skeleton, color, and depth) from each Kinect device in the system. These three streams of camera data will be used to translate user movement data into raw camera angles that can be converted into Unity Pro camera actions. The output of the Kinect SDK module of the system will be then modified by a wrapper class to correctly format the data for use in the Unity Pro module.

3.1.2 Local Class Diagram (UML)

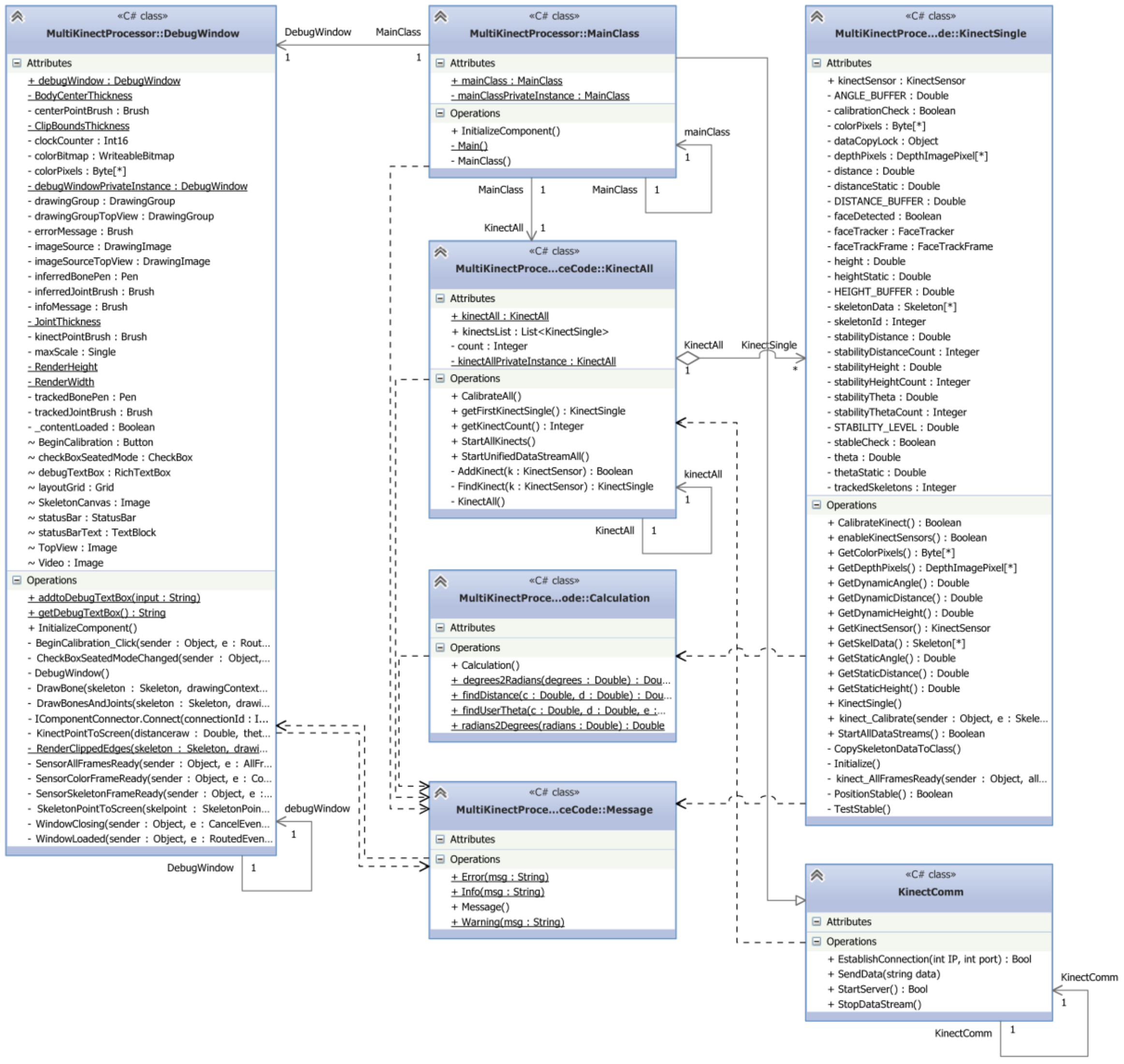

«C# class»
MultiKinectProcessor::DebugWindow
Attributes
+ debugWindow : DebugWindow
- BodyCenterThickness
- centerPointBrush : Brush
- ClipBoundsThickness
- clockCounter : Int16
- colorBitmap : WriteableBitmap
- colorPixels : Byte[*]
- debugWindowPrivateInstance : DebugWindow
- drawingGroup : DrawingGroup
- drawingGroupTopView : DrawingGroup
- errorMessage : Brush
- imageSource : DrawingImage
- imageSourceTopView : DrawingImage
- inferredBonePen : Pen
- inferredJointBrush : Brush
- infoMessage : Brush
- JointThickness
- kinectPointBrush : Brush
- maxScale : Single
- RenderHeight
- RenderWidth
- trackedBonePen : Pen
- trackedJointBrush : Brush
- _contentLoaded : Boolean
~ BeginCalibration : Button
~ checkBoxSeatedMode : CheckBox
~ debugTextBox : RichTextBox
~ layoutGrid : Grid
~ SkeletonCanvas : Image
~ statusBar : StatusBar
~ statusBarText : TextBlock
~ TopView : Image
~ Video : Image
Operations
+ addtoDebugTextBox(input : String)
+ getDebugTextBox() : String
+ InitializeComponent()
- BeginCalibration_Click(sender : Object, e : Rout...
- CheckBoxSeatedModeChanged(sender : Object,...
- DebugWindow()
- DrawBone(skeleton : Skeleton, drawingContext...
- DrawBonesAndJoints(skeleton : Skeleton, drawi...
- IComponentConnector.Connect(connectionId : I...
- KinectPointToScreen(distanceraw : Double, thet...
- RenderClippedEdges(skeleton : Skeleton, drawi...
- SensorAllFramesReady(sender : Object, e : AllFr...
- SensorColorFrameReady(sender : Object, e : Co...
- SensorSkeletonFrameReady(sender : Object, e :...
- SkeletonPointToScreen(skelpoint : SkeletonPoin...
- WindowClosing(sender : Object, e : CancelEven...
- WindowLoaded(sender : Object, e : RoutedEven...
DebugWindow
MainClass
1
1
«C# class»
MultiKinectProcessor::MainClass
Attributes
+ mainClass : MainClass
- mainClassPrivateInstance : MainClass
Operations
+ InitializeComponent()
- Main()
- MainClass()
mainClass
1
MainClass
1
MainClass
1
KinectAll
1
«C# class»
MultiKinectProce...ceCode::KinectAll
Attributes
+ kinectAll : KinectAll
+ kinectsList : List<KinectSingle>
- count : Integer
- kinectAllPrivateInstance : KinectAll
Operations
+ CalibrateAll()
+ getFirstKinectSingle() : KinectSingle
+ getKinectCount() : Integer
+ StartAllKinects()
+ StartUnifiedDataStreamAll()
- AddKinect(k : KinectSensor) : Boolean
- FindKinect(k : KinectSensor) : KinectSingle
- KinectAll()
KinectAll
KinectSingle
1
*
kinectAll
1
KinectAll
1
«C# class»
MultiKinectProce...ode::Calculation
Attributes
Operations
+ Calculation()
+ degrees2Radians(degrees : Double) : Dou...
+ findDistance(c : Double, d : Double) : Dou...
+ findUserTheta(c : Double, d : Double, e :...
+ radians2Degrees(radians : Double) : Double
«C# class»
MultiKinectProce...ceCode::Message
Attributes
Operations
+ Error(msg : String)
+ Info(msg : String)
+ Message()
+ Warning(msg : String)
debugWindow
1
DebugWindow
1
«C# class»
MultiKinectProce...de::KinectSingle
Attributes
+ kinectSensor : KinectSensor
- ANGLE_BUFFER : Double
- calibrationCheck : Boolean
- colorPixels : Byte[*]
- dataCopyLock : Object
- depthPixels : DepthImagePixel[*]
- distance : Double
- distanceStatic : Double
- DISTANCE_BUFFER : Double
- faceDetected : Boolean
- faceTracker : FaceTracker
- faceTrackFrame : FaceTrackFrame
- height : Double
- heightStatic : Double
- HEIGHT_BUFFER : Double
- skeletonData : Skeleton[*]
- skeletonId : Integer
- stabilityDistance : Double
- stabilityDistanceCount : Integer
- stabilityHeight : Double
- stabilityHeightCount : Integer
- stabilityTheta : Double
- stabilityThetaCount : Integer
- STABILITY_LEVEL : Double
- stableCheck : Boolean
- theta : Double
- thetaStatic : Double
- trackedSkeletons : Integer
Operations
+ CalibrateKinect() : Boolean
+ enableKinectSensors() : Boolean
+ GetColorPixels() : Byte[*]
+ GetDepthPixels() : DepthImagePixel[*]
+ GetDynamicAngle() : Double
+ GetDynamicDistance() : Double
+ GetDynamicHeight() : Double
+ GetKinectSensor() : KinectSensor
+ GetSkelData() : Skeleton[*]
+ GetStaticAngle() : Double
+ GetStaticDistance() : Double
+ GetStaticHeight() : Double
+ KinectSingle()
+ kinect_Calibrate(sender : Object, e : Skele...
+ StartAllDataStreams() : Boolean
- CopySkeletonDataToClass()
- Initialize()
- kinect_AllFramesReady(sender : Object, all...
- PositionStable() : Boolean
- TestStable()
«C# class»
KinectComm
Attributes
Operations
+ EstablishConnection(int IP, int port) : Bool
+ SendData(string data)
+ StartServer() : Bool
+ StopDataStream()
KinectComm
1
KinectComm
1

Figure 7. Kinect SDK Module UML Class Diagram

3.1.3 Class Data and Method Descriptions

3.1.3.1 StartClass

The `StartClass` is the class that serves as a main starting point when calling the Kinect SDK Module independent of the wrapper or Unity modules. This class contains support function that allow the debug window to appear and be able to test and debug the Kinect SDK module output without any Unity 3D renderings

3.1.3.1.1 Class Properties

Singleton Class

3.1.3.1.2 Private Member Variables

`private static StartClass mainClassPrivateInstance`
The self-instantiated single instance of `StartClass`

3.1.3.1.3 Private Member Methods

`private StartClass()`
Private constructor method ensures that this class is never created again in program runtime

3.1.3.1.4 Public Member Methods

`public static StartClass mainClass()`
Public getter function returns the private instance of main class.

3.1.3.2 KinectAll

The `KinectAll` class is a singleton class that contains Kinect system wide parameters. This includes a `KinectSingle` list and a count of number of connected Kinects.

3.1.3.2.1 Class Properties

Singleton Class

3.1.3.2.2 Private Member Variables

`private static KinectAll kinectAllPrivateInstance`
The self-instantiated single instance of `KinectAll`.

`private List<KinectSingle> kinectsList`
A list of all the Kinects that is connected to the system.

3.1.3.2.3 Private Member Methods

`private KinectAll()`
Constructor for `KinectAll` class. The constructor creates the single instance of the `KinectAll` class.

`private bool AddKinect(KinectSensor k)`
Adds `KinectSensor k` to `List<KinectSingle> kinectsList`. This function also checks whether `KinectSensor k` has already been previously added.

Parameters
`KinectSensor k` - A newly detected Kinect that needs to be added to `List<KinectSingle> kinectsList`

Returns

`Boolean (True/False)` - Return True to indicate that `KinectSensor k` has been successfully added to `List<KinectSingle> kinectsList`. Return False to indicate `KinectSensor k` has been unsuccessfully added to `List<KinectSingle> kinectsList`.

`private KinectSingle FindKinect(KinectSensor k)`
This function determines whether `KinectSensor k` is in `List<KinectSingle> kinectsList`.

Parameters
`KinectSensor k` - The kinect the function tries to find in `List<KinectSingle>`

Returns
`Boolean (True/False)` - Return True to indicate that `KinectSensor k` has been successfully found in `List<KinectSingle> kinectsList`. Return False to indicate `KinectSensor k` has not been found in `List<KinectSingle> kinectsList`.

3.1.3.2.4 Public Member Methods

`public void StartAllKinects()`
This function detects all connect Kinects, populates a list of all the connected Kinects and starts/enables them.

`public void CalibrateAll()`
Calibrates all the Kinects detected in the system.

`public void StartUnifiedDataStreamAll()`
Starts all the need data streams for each Kinect

3.1.3.3 KinectSingle

3.1.3.3.1 Class Properties

No special properties

3.1.3.3.2 Private Member Variables

`private Object dataCopyLock`
Thread lock object that ensures whatever code encapsulated in a `lock{}` is allowed to complete undisrupted.

`private double distance`
Holds the distance of the user from the Kinect sensor in meters (dynamic). This value constantly gets updated regardless of calibration done.

`private double distanceStatic`
Holds the distance of the user from the Kinect at the completion of calibration sequence.

`private double height`
Holds the height of the user relative to the Kinect (dynamic). This value constantly gets updated regardless of calibration done.

`private double heightStatic`
Holds the height of the user relative to the Kinect at completion of calibration sequence.

`private double theta`
Holds the dynamic angle of the Kinect relative to the user (dynamic). This value constantly gets updated regardless of calibration done.

`private double thetaStatic`

Holds the static angle of Kinect relative to user obtained at completion of calibration sequence.

`private Skeleton[] skeletonData`
Holds an array of `Skeleton` objects that is copied from the `KinectSensor` at each frame. Each `Skeleton` item represents an entire body, each body will receive one index in this array

`private int skeletonId`
Holds unique skeleton ID of the tracked individual

`private bool calibrationCheck`
Is true if calibration has been successfully performed

`private int stabilityDistanceCount`
Counts the number of frames that the distance value is in stable state.

`private int stabilityThetaCount`
Counts the number of frames that the theta value is in stable state.

`private int stabilityHeightCount`
Counts the number of frames that the height value is in stable state.

`private double stabilityTheta`
Used during calibration to hold the base theta value for stability comparison until the calibration process is successfully completed.

`private double stabilityDistance`
Used during calibration to hold the base distance value for stability comparison until the calibration process is successfully completed.

`private double stabilityHeight`
Used during calibration to hold the base height value for stability comparison until the calibration process is successfully completed.

`readonly private double STABILITY_LEVEL`
Parameter that is used during calibration. There must by `STABILITY_LEVEL` frames elapsed in stable position before calibration values are stored and declared calibrated

`readonly private double DISTANCE_BUFFER`
Parameter that is used during calibration. The distance +/- range that the person can move to still be considered stable.

`readonly private double HEIGHT_BUFFER`
Parameter that is used during calibration. The height +/- range that the person can move to still be considered stable.

`readonly private double ANGLE_BUFFER`
Parameter that is used during calibration. The theta +/- range that the person can move to still be considered stable.

`private FaceTracker faceTracker`
Face tracker class instance that contains the `Track()` function, enabling user face detection.

`private FaceTrackFrame faceTrackFrame`
Holds face tracking data from each frame, also contains `public bool TrackSuccessful()`

`private bool faceDetected`
True when a face is detected in the latest frame received from `KinectSensor`

`private byte[] colorPixels`
Holds color stream pixel data in the latest frame received from `KinectSensor`

`private short[] depthPixels`

Holds depth stream pixel data in the latest frame received from `KinectSensor`

`private Skeleton skeletonOfInterest`
Main Skeleton that is used to control the entire Unity 3D scene. Passed into the `Track()` function inside `faceTracker`

3.1.3.3.3 Private Member Methods

`private void Initialize()`
Initializes all stability variables to zero to prepare for calibration and initializes `stableCheck` and `calibrationCheck` to `false`

`private bool PositionStable()`
Checks that skeleton position is stable

`private void kinect_Calibrate(object sender, SkeletonFrameReadyEventArgs e)`
Event handler called at each ready skeleton data frame to initiate the stability check method. When counters have determined stability is reached, values are stored and calibration is finished.

`private void kinect_AllFramesReady(object sender, AllFramesReadyEventArgs allFramesReadyEventArgs)`
Event handler called each time all 3 data streams have changed (30fps). This handler copies all of the KinectSensor data to data structures in KinectSingle class to begin processing. Processing includes detecting if the person is facing to or away from the Kinect and then calculating theta and distance for each Kinect. The handler begins with locking a semaphore to prevent the camera from taking any data while data is being processed. At the end of the sequence, the semaphore is released.

3.1.3.3.4 Public Member Methods

`public KinectSingle()`
Constructor of this class is public, as many instances of this class may be created as long as it is added to the `List<KinectSingle> kinectsList` list. Calls `Initialize()` to make sure all calibration values are reset and Kinect is added in uncalibrated mode.

`public bool enableKinectSensors()`
Starts Kinect Sensor by enabling skeleton, color, and depth streams

Returns
True if Kinect Sensor is successfully enabled

`public bool StartAllDataStreams()`
Links `kinect_AllFramesReady` as thr `AllFramesReadyEventArgs` event handler. Initializes `skeletonData`, `colorPixels`, and `depthPixels` arrays to size of their respective data lengths for each frame

Returns
True if successfully completed

`public bool CalibrateKinect()`
Sets up event handler as `kinect_Calibrate` every time a new Skeleton Frame is received from the Kinect Sensor. Calls `PositionStable()` to hold the thread open until calibration completes. Disconnects event handler upon calibration success.

Returns:
True if successful completed

3.1.3.4 Calculation

3.1.3.4.1 Class Properties

Static, class constructor is private and never ever gets instantiated

3.1.3.4.2 Public Methods

`static public double findDistance(double c, double d)`
Find the distance from coordinate (0, 0) to coordinate (c, d) by using the distance formula.

Parameters
`double c` - X coordinate
`double d` - Y coordinate

Returns
Distance (in Meters) given the coordinate for a user's center point

`static public double findUserTheta(double c, double d, double e, double f)`
Finds the angle of the Kinect location relative to the user starting at the positive y axis being 0 degrees, using the law of Cosine

Parameters
`double c`
`double d`
`double e`
`double f`

Returns
A double value of angle of the Kinect location relative to the user

`static public double degrees2Radians(double degrees)`
Converts degree value to radian value

`static public double radians2Degrees(double radians)`
Converts radian value to degree value

3.1.3.5 Message

3.1.3.5.1 Class Properties

Static, class constructor is private and never ever gets instantiated

3.1.3.5.2 Public Methods

`static void Error(String msg)`
Adds `String msg` to the debug textbox in the debug console UI and prints to Debug Console with Error message type. Error message types will append "`ERROR:`" to the beginning of the string and color it red where rich text is supported

`static void Warning(String msg)`
Adds `String msg` to the debug textbox in the debug console UI and prints to Debug Console with Warning message type. Warning message types will append "`WARNING:`" to the beginning of the string and color it yellow/orange where rich text is supported

`static void Info(String msg)`
Adds `String msg` to the debug textbox in the debug console UI and prints to Debug Console with Info message type. Info message types have standard black text formatting.

3.1.3.6 KinectComm

3.1.3.6.1 Overview

This class contains the code used for communications between the Kinect module and the Unity module. The Kinect module serves as a server while the Unity module serves as a client

3.1.3.6.2 Network Time Sequence Diagram

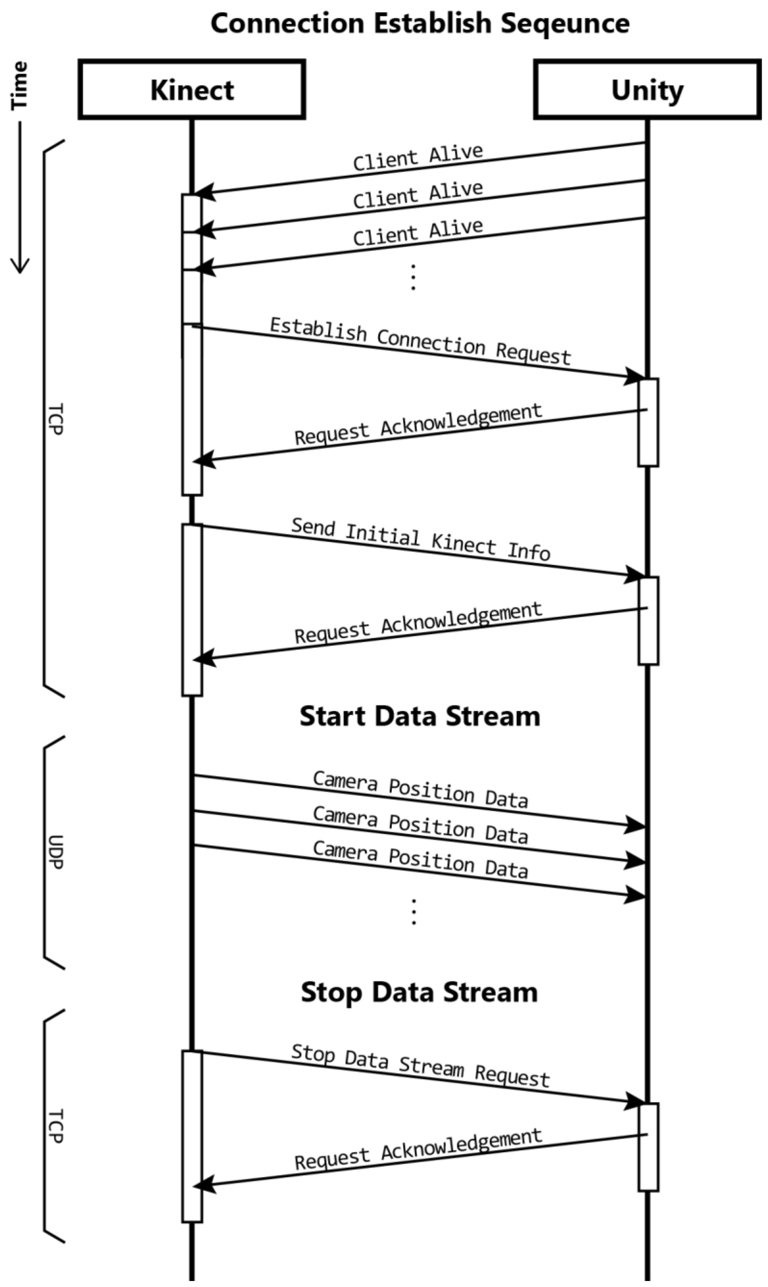


Figure 8. Network Connection Time Sequence Diagram

3.1.3.6.3 Public Member Methods

`public bool StartServer()`

Starts the server for the kinect module. Binds listening ports.

Return:

Boolean that states whether the server was started successfully or not. Return true if server is started sucessfully. Return false if server was not able to start.

`public bool EstablishConnection(int IP, int port)`
Establishes the connection parameters between the Kinect Module and Unity Module

Parameters:
IP address of Unity module and ports its listening on.

Returns:
Boolean that states whether connection between Kinect and Unity module has been successfully established.

`public void SendData(string data)`
Sends a data stream over UDP

Parameters:
String data that needs to be sent from Kinect to Unity

`public bool StopDataStream()`
Sends a stop signal from Kinect Module to Unity.

Returns:
A boolean stating whether the stop signal has been successfully transmitted.

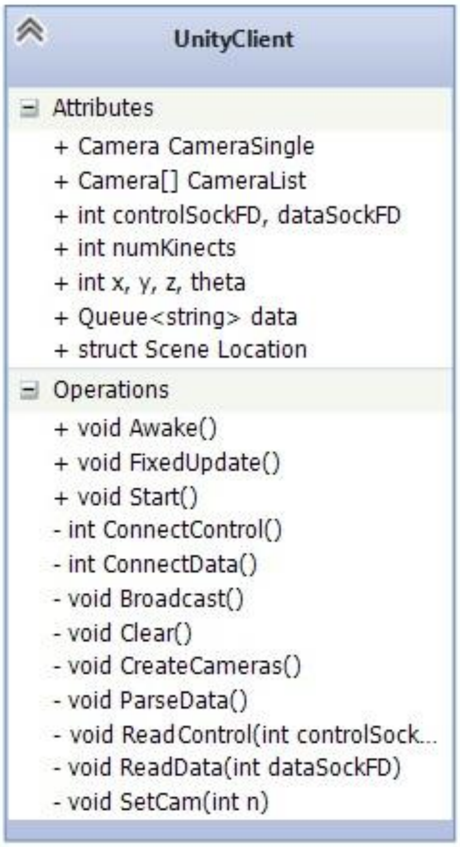
3.1.3.7 DebugWindow

3.2 Unity Front End Code

3.2.1 Overview

The Unity3D sector of the Program acts as the Client. The Unity Client begins with a Broadcast message sent over TCP to tell the Kinect Server the Client's IP address. When the Kinect Server is done with its calibration phase, it establishes a connection to Unity. The Unity Client then receives Control info (# of Kinects, Position, Location (Default if first time)), creates the Camera objects, and sets initial user position and scene location. After this phase of Unity calibration, an ACK is sent to the Server and the Server responds with a message to start the data stream. The data stream connection is established and Unity is now waiting for Kinect User Input data. In this state, Unity is constantly receiving and using data to portray the User's view of the window. If the scene location is moved in Unity, the location is saved in Unity and also sent to the Kinect Server. If the Kinect Server resets to adjust for more or less Kinects, it sends a Stop message to the Unity Client. Unity then handles this reset and clears all Cameras. The process returns to when the Unity Client receives Control info. If the Kinect Server or the Unity Client crashes, this whole process restarts with Unity sending a Broadcast message.

3.2.2 Local Class Diagram (UML)



UnityClient

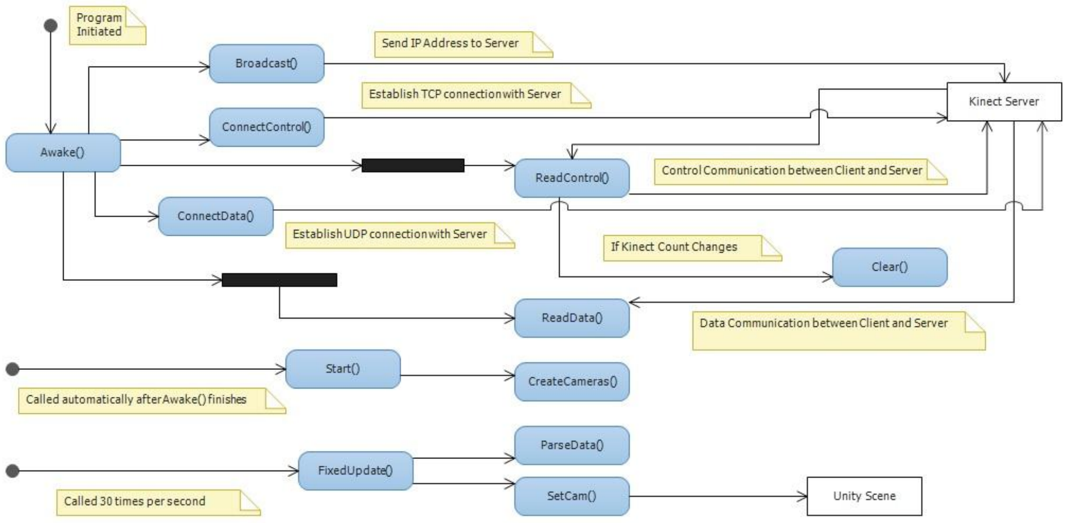
Attributes

```
+ Camera CameraSingle
+ Camera[] CameraList
+ int controlSockFD, dataSockFD
+ int numKinects
+ int x, y, z, theta
+ Queue<string> data
+ struct Scene Location
```

Operations

```
+ void Awake()
+ void FixedUpdate()
+ void Start()
- int ConnectControl()
- int ConnectData()
- void Broadcast()
- void Clear()
- void CreateCameras()
- void ParseData()
- void ReadControl(int controlSock...
- void ReadData(int dataSockFD)
- void SetCam(int n)
```

3.2.3 Local Class Activity Diagram


Program Initiated
Broadcast()
Send IP Address to Server
Establish TCP connection with Server
Kinect Server
ConnectControl()
Awake()
ReadControl()
Control Communication between Client and Server
ConnectData()
Establish UDP connection with Server
If Kinect Count Changes
Clear()
ReadData()
Data Communication between Client and Server
Start()
CreateCameras()
Called automatically after Awake() finishes
ParseData()
FixedUpdate()
SetCam()
Unity Scene
Called 30 times per second

3.2.4 Class Data and Method Descriptions

3.2.4.1 UnityClient Script (Main() Class)

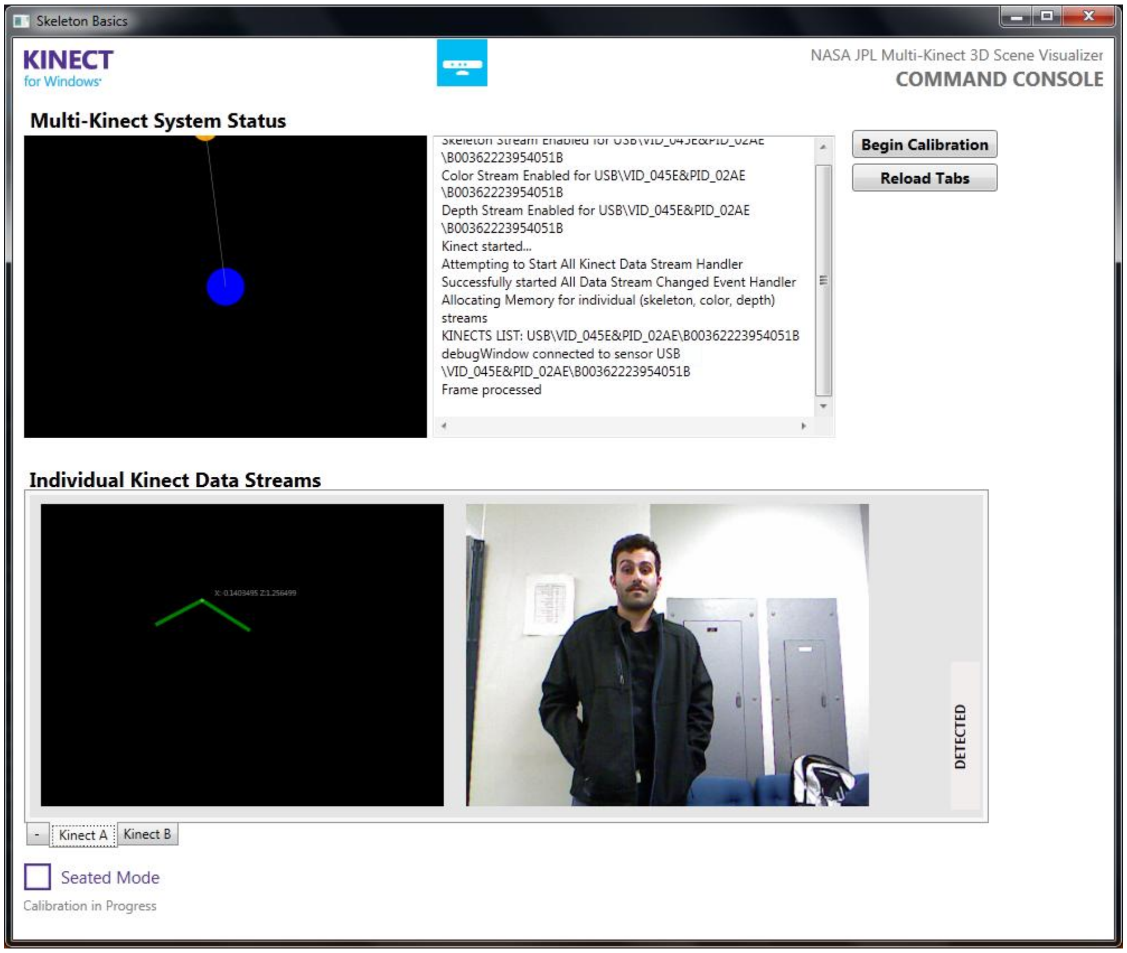
```
public Camera[] CameraList = NULL;
public Queue<string> data = new Queue<string>();
public Camera CameraSingle;
public int controlSockFD, dataSockFD;
public int x, y, z, theta; //user location coordinates and angle
public int numKinects;


public struct SceneLocation
     //variables to store location of Scene in Unity


public void Awake()
     Calls Broadcast()
     Call ConnectControl()
     Spin off new thread ReadControl() to handle TCP Control Stream
     Call ConnectData()
Spin off new thread ReadData() to handle UDP Data Stream
     When function is completed, Start() is automatically called
private void Broadcast()
     Broadcasts IP address for Kinect Server to find
private int ConnectControl()
     Wait for TCP connection initiation packet
     Receive packet
     Sends ACK
Wait for Control info
     Receive Number of Kinects, User Position, and Scene Location (Default if first time)
     Updates Global Variables
     Sends ACK
     return controlSockFD
private void ReadControl(int controlSockFD)
     Constantly receiving control messages over TCP
     If scene location moved
             Update SceneLocation Struct variables
             send SceneLocation Struct variables to Kinect Server
     If Control message received
             Call Clear()
private int ConnectData()
     Wait for UDP connection initiation packet
     Receive packet
     Sends ACK (Connection established)
     Waiting for User Input Data
     return dataSockFD
private void ReadData(int dataSockFD)
     Constantly receiving data over UDP
Constantly storing packet info in the data queue
public void Start()
     CameraSingle = Camera.Main
     If number of Kinects > 1
Calls CreateCameras()
private void CreateCameras()
     For each additional KinectSingle in KinectAll.kinectList
Create new Camera object in Unity3D scene
Adds Camera object to CameraList
Set initial user position
Set scene location (Default if first time)
public void FixedUpdate() //automatically called 30 times per second
     Calls ParseData()
     Calls SetCam(int n)
```

```
private void ParseData()
     Reads data from the data queue
     Parses the data so that it can be used in SetCam()
private void SetCam(int n)
     Sets Camera n’s position in the scene based on the x,y,z,theta values from ParseData()
private void Clear()
     Destroy all Camera Objects except CameraSingle
     Clear CameraList
     Clear the data queue
```

3.3 Front End User Interface

3.3.1 Overview

3.3.2 Mock Up Diagrams



```
Skeleton Stream Enabled for USB\VID_045E&PID_02AE
\B00362223954051B
Color Stream Enabled for USB\VID_045E&PID_02AE
\B00362223954051B
Depth Stream Enabled for USB\VID_045E&PID_02AE
\B00362223954051B
Kinect started...
Attempting to Start All Kinect Data Stream Handler
Successfully started All Data Stream Changed Event Handler
Allocating Memory for individual (skeleton, color, depth)
streams
KINECTS LIST: USB\VID_045E&PID_02AE\B00362223954051B
debugWindow connected to sensor USB
\VID_045E&PID_02AE\B00362223954051B
Frame processed
```

Figure 9. Command Console UI Mockup

4 Conclusion

Work In Progress here

Appendix A. Glossary

3D – Three Dimensional

API – See *Application Programming Interface*

Application Programming Interface – A set of source code with a protocol intended for software packages to communicate with each other. It may also contain documentation that describes its code structure.

Attribute – A structure item of a class. Equivalent to *member variable*, *property*, *data member*, or *field*.

Boolean – a 1-bit variable type that is either 1 (true) or 0 (false)

FPS – Frames Per Second

IDE – See *Integrated Development Environment*

Integrated Development Environment – An application tool suite which assists in developing code

JPL – Jet Propulsion Laboratory

Kinect SDK – The libraries provided by Microsoft to communicate and effectively use data received from the Kinect

Method – A behavior item of a class. It is the standard naming of a function that is defined within a class and is essentially a member of it. A method has all access to private and public data members of its class.

NASA – National Aeronautics and Space Administration

SDK – Software Development Kit

Singleton Class – A class that is instantiated once and only once at program startup. Constructors are of private type to prevent the class from being instantiated again elsewhere. No member function should take advantage of this private constructor and call it any time throughout the program. The initial instance is of private static type and a public static getter function is instantiated for public access to this singleton class instance.

SRS – Software Requirement Specification

UCSB – University of California at Santa Barbara

UML – See *Universal Modeling Language*

Unity 3D – the rendering engine and IDE where the panoramic scene is generated and visualized. See *IDE*

Universal Modeling Language – standardized visual modeling language used for representing object-oriented programming.

Visual Studio – An IDE created by Microsoft used for developing Windows applications

Workspace – The area in which the user stands

Appendix B. Index