Vuforia-Wall Manual

https://github.com/Duffycola/Vuforia-Wall
Author: Eduard Feicho

About

Vuforia-Wall is a Virtual Reality iPhone App based on Vuforia-SDK from Qualcomm.
The simple idea is to provide a virtual image wall. Pick images from your iPhone camera or photo library, arrange them and watch them in Augmented Reality.

Installation

XCode IDE and Apple Developer Program
Make sure XCode is installed (https://developer.apple.com/xcode/) or install from the Mac App Store. To build Apps on the device, membership in the Apple iPhone Developer Program is required ($99/year).

Qualcomm Vuforia SDK
Vuforia-Wall requires Qualcomm Vuforia SDK for iOS available from "https://ar.qualcomm.at/qdevnet/sdk/ios". Download and install the SDK (**vuforia-sdk-ios-1-5-8.zip**). As of date, the working version is 1-5-8.
Per default the SDK is installed to ~/Development/iOS/vuforia-sdk-ios-1-5-8/.

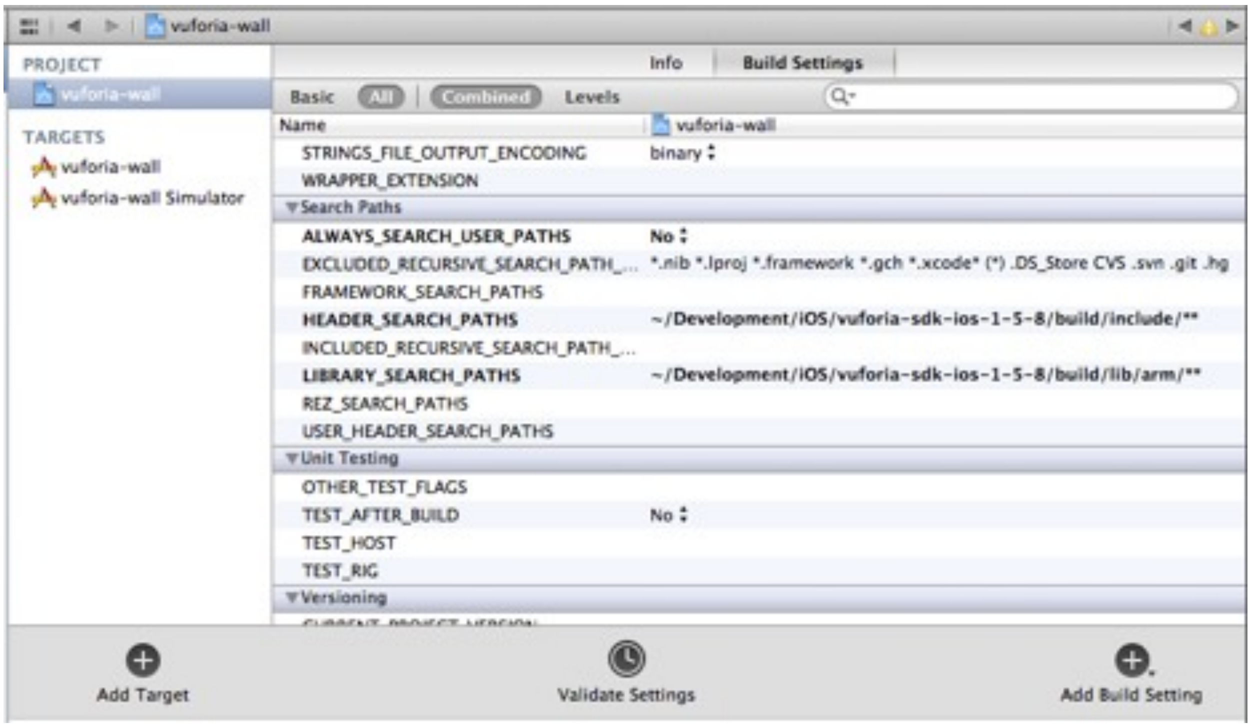
To be sure everything works correct with your installation, try to compile the samples in ~/Development/iOS/vuforia-sdk-ios-1-5-8/samples/.

You might get an error like "no valid provisioning device found". The problem is, in order to compile for devices, you need to have a valid Developer Certificate from the Apple Developer Program. Thus, without paying for the membership, you won't be able to simulate the Vuforia-SDK, as the Vuforia library only contains symbols for iOS devices, but does not work only with the iPhone Simulator.

Also note, that the ARC feature (Automatic Reference Counting) is quite new and the Vuforia SDK doesn't compile with ARC. When using ARC, you have to fix retain, release, and NSAutoReleasePool calls, as well as some retain properties to weak/strong. Therefore, prefer to disable ARC when you do a new project.

Vuforia-Wall Xcode Project
Download and open Vuforia-Wall Xcode Project from github (https://github.com/Duffycola/Vuforia-Wall). Alternatively, checkout the git repository using git:

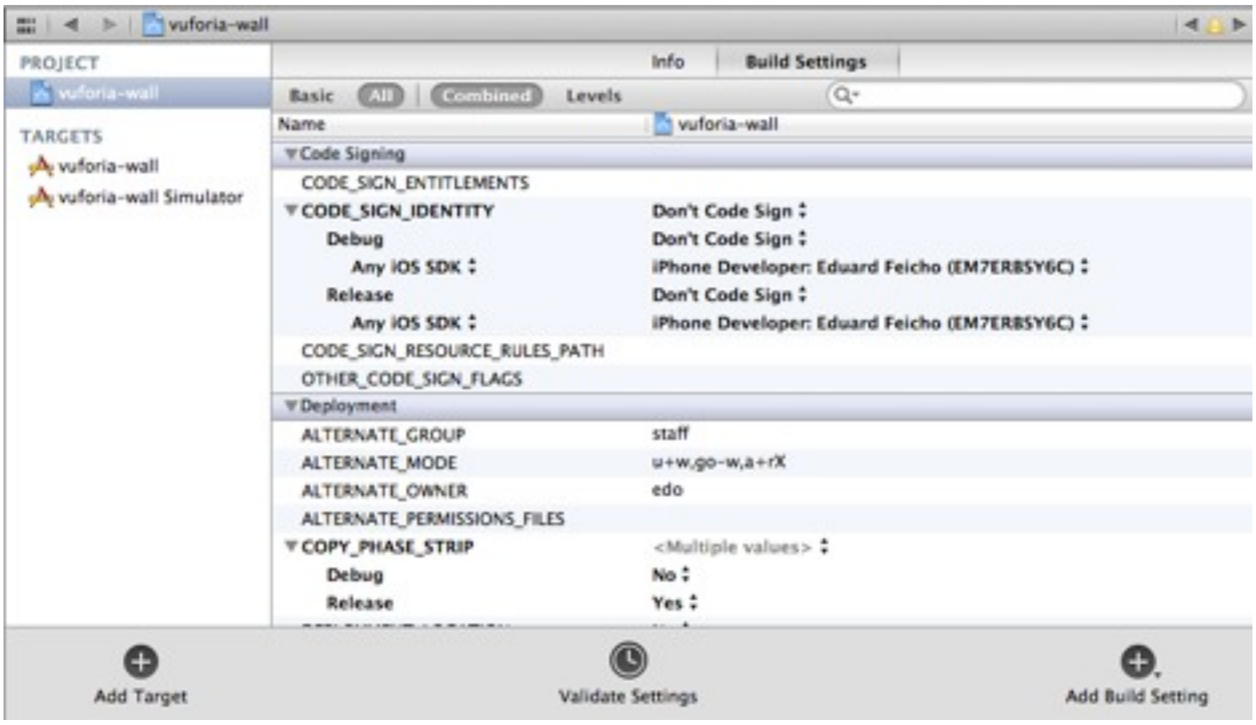
```
$ mkdir vuforia-wall
cd vuforia-wall
git init
git remote add origin https://github.com/Duffycola/Vuforia-Wall.git
git pull
```



Name	vuforia-wall
STRINGS_FILE_OUTPUT_ENCODING	binary
WRAPPER_EXTENSION	
Search Paths	
ALWAYS_SEARCH_USER_PATHS	No
EXCLUDED_RECURSIVE_SEARCH_PATH_...	*.nib *.lproj *.framework *.gch *.xcode* (*) .DS_Store CVS .svn .git .hg
FRAMEWORK_SEARCH_PATHS	
HEADER_SEARCH_PATHS	~/Development/iOS/vuforia-sdk-ios-1-5-8/build/include/**
INCLUDED_RECURSIVE_SEARCH_PATH_...	
LIBRARY_SEARCH_PATHS	~/Development/iOS/vuforia-sdk-ios-1-5-8/build/lib/arm/**
REZ_SEARCH_PATHS	
USER_HEADER_SEARCH_PATHS	
Unit Testing	
OTHER_TEST_FLAGS	
TEST_AFTER_BUILD	No
TEST_HOST	
TEST_RIG	
Versioning	

open vuforia-wall.xcodeproj

Possibly, the library paths are not correct. If libQCAR.a appears red (ext/Vuforia-SDK/libQCAR.a) try to remove it from the project and manually add it again per Drag-and-Drop



Name	vuforia-wall
Code Signing	
CODE_SIGN_ENTITLEMENTS	
CODE_SIGN_IDENTITY	Don't Code Sign
Debug	Don't Code Sign
Any iOS SDK	iPhone Developer: Eduard Feicho (EM7ER85Y6C)
Release	Don't Code Sign
Any iOS SDK	iPhone Developer: Eduard Feicho (EM7ER85Y6C)
CODE_SIGN_RESOURCE_RULES_PATH	
OTHER_CODE_SIGN_FLAGS	
Deployment	
ALTERNATE_GROUP	staff
ALTERNATE_MODE	u+w,go-w,a+rX
ALTERNATE_OWNER	edo
ALTERNATE_PERMISSIONS_FILES	
COPY_PHASE_STRIP	<Multiple values>
Debug	No
Release	Yes

located at ~/Development/iOS/vuforia-sdk-ios-1-5-8/build/lib/arm/libQCAR.a.

Furthermore, make sure the project settings are similar to the following pictures. More precisely,

• HEADER_SEARCH_PATHS points to ~/Development/iOS/vuforia-sdk-ios-1-5-8/build/include/

• LIBRARY_SEARCH_PATHS points to ~/Development/iOS/vuforia-sdk-ios-1-5-8/build/lib/arm/
When you have installed a developer certificate correctly following the instructions in from the iPhone developer program website, CODE_SIGN_IDENTITY should point to your certificate, like in this screenshot.

Note, the project has a simulator target, but it will build without Vuforia SDK and thus Augmented Reality view will not be visible. This is useful for testing SDK unrelated code.

Implementation

App Layout

The App is based on one of the sample projects and is a simple TabBar based application. Vuforia SDK already provides a ARParentViewController to use as a ViewController for the Augmented Reality View. Thus, in AppDelegate.m the view is simply initialized:

AppDelegate.m

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions;
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    // Override point for customization after application launch.

     [[UIApplication sharedApplication] setStatusBarHidden:YES
withAnimation:UIStatusBarAnimationNone];

     UIViewController *vc1, *vc2, *vc3;

     vc1 = [[ImagePickerViewController alloc] init];
     vc2 = [[ImageWallViewController alloc] init];
#if !(TARGET_IPHONE_SIMULATOR)
     vc3 = [[QCARViewController alloc] init];
#endif

     self.tabBarController = [[UITabBarController alloc] init];
     self.tabBarController.delegate = self;

#if !(TARGET_IPHONE_SIMULATOR)
     self.tabBarController.viewControllers = [NSArray arrayWithObjects:vc1,vc2,vc3,nil];
#else
     self.tabBarController.viewControllers = [NSArray arrayWithObjects:vc1,vc2,nil];
#endif

     self.window.rootViewController = self.tabBarController;
    [self.window makeKeyAndVisible];
    return YES;
}
```

ImageWall Model

Picking images from the library is done using the iOS standard library. For more information, look for Apple documentation on "UIImagePickerController" (http://developer.apple.com/library/ios/#DOCUMENTATION/UIKit/Reference/UIImagePickerController_Class/UIImagePickerController/UIImagePickerController.html).

At the core of the application the images are stored in a singleton container called ImageWall, which wraps the images in a mutable array. The UIImagePickerController implementation and the model are losely-coupled: ImageWall acts as observer to NSNotificationCenter waiting for events to add (UIImagePicker) or remove (user operation) images:

ImageWall.h

```
static NSString* notificationImageWallAddImage = @"ImageWallAddImageNotification";
static NSString* notificationImageWallRemoveImage = @"ImageWallRemoveImageNotification";
```

ImageWall.m

```
- (void)actionImagePicked:(NSNotification*)notification;
{
     if (notification.object == nil) {
             return;
     }

     UIImage* image = notification.object;
     CGSize imageSize = [image size];
     imageSize = CGSizeMake(imageSize.width / 4.0, imageSize.height / 4.0);
     CGRect imageFrame = CGRectMake((self.frame.size.width-imageSize.width)/2.0,
(self.frame.size.height-imageSize.height)/2.0, imageSize.width, imageSize.height);

     TouchImageView* touchImage = [[TouchImageView alloc] initWithFrame:imageFrame];
     touchImage.image = image;

     [self.images addObject:touchImage];

     [[NSNotificationCenter defaultCenter]
postNotificationName:notificationImageWallAddImage object:touchImage];
}

- (void)actionImageRemoved:(NSNotification*)notification;
{
     if (!notification.object) {
             return;
     }

     TouchImageView* touchImage = notification.object;
     [self.images removeObject:touchImage];

     [[NSNotificationCenter defaultCenter]
postNotificationName:notificationImageWallRemoveImage object:touchImage];
}
```

Gestures
Standard gestures in newer versions of iOS are pretty easy to implement using GestureRecognizer. For more information read Event Handling Guide for iOS, chapter Gesture Recognizers (http://developer.apple.com/library/ios/#documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/GestureRecognizers/GestureRecognizers.html).

The class TouchImageView is a custom UIImageView class with gesture support to pan, pinch and rotate an ImageView. Using GestureRecognizer, implementation is fairly simple. For example supporting a rotation Gesture:

TouchImageView.m

```
- (void)createGestureRecognizers;
{
     UIRotationGestureRecognizer *rotationGesture = [[UIRotationGestureRecognizer alloc]
             initWithTarget:self action:@selector(handleRotationGesture:)];
     [self addGestureRecognizer:rotationGesture];
}
- (IBAction)handleRotationGesture:(UIRotationGestureRecognizer *)sender;
{
     drotation = [sender rotation] * 50.0;
     [self updateImageTransform];
     if (sender.state == UIGestureRecognizerStateEnded) {
             rotation = rotation + drotation;
     }
     drotation = 0.0;
}
```

Note, some gesture recognizers send continuous values relative to the current gesture, such as with rotation. Thus, a current rotation value is tracked as well as a delta value. When the UIGestureRecognizerState ends, the current value is updated. In updateImageTransform all the values for translation, rotation and scale are concatenated into a single CGAffineTransform:

TouchImageView.m

```
- (void)updateImageTransform;
{
     float x_new = x + dx;
     float y_new = y + dy;
     float rotation_new = rotation + drotation;
     float scale_new = scale * dscale;

     CGAffineTransform t_translate = CGAffineTransformMakeTranslation(x_new, y_new);
     CGAffineTransform t_rotation = CGAffineTransformMakeRotation(rotation_new / 180.0 *
3.14); // degree to radian
     CGAffineTransform t_scale = CGAffineTransformMakeScale(scale_new, scale_new);

     self.transform = CGAffineTransformConcat(CGAffineTransformConcat(t_translate,
t_rotation), t_scale);
}
```

Most importantly, don't forget your UIView to support multiple touches:

TouchImageView.m

```
     self.userInteractionEnabled = YES;
     self.multipleTouchEnabled = YES;
```

Custom Rendering

Note, the iPhone/iOS and Vuforia SDK support OpenGL ES. To finally render custom data EAGLView.mm is customized. Note, the extension *.mm means that the file contains mixed code, both Objective-C and C. First, the SDK initializes its 3D objects:

```
- (void)setup3dObjects
{
    // build the array of objects we want drawn and their texture
    // in our app, the textures are not static files but comes from uiimage data.

     NSLog(@"EAGLView setup3dObjects");

     [objects3D removeAllObjects];
    for (int i=0; i < [ImageWall sharedInstance].images.count; i++) {
TouchImageView* imageView = [[ImageWall sharedInstance].images
objectAtIndex:i];
Plane3D *obj3D = [[Plane3D alloc] init];

obj3D.dx = [imageView myX];
obj3D.dy = [imageView myY];
obj3D.rotation = [imageView myRotation];
obj3D.scale = [imageView myScale];

[obj3D setTextureWithImage:imageView.image];

[objects3D addObject:obj3D];
    }
}
```

Here, we set up custom 3D planes (Object3D is a class from Vuforia SDK but Plane3D is a subclass that was implemented manually) and provide bind the picked image data as texture to OpenGL. Then, QCAR calls the main rendering method that can to be customized:

```
// *** QCAR will call this method on a single background thread ***
- (void)renderFrameQCAR;
{
    [self setFramebuffer];

    // Clear colour and depth buffers
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Render video background and retrieve tracking state
    QCAR::State state = QCAR::Renderer::getInstance().begin();
    QCAR::Renderer::getInstance().drawVideoBackground();

    // Enable certain GL states
    ...

    if (state.getNumActiveTrackables() > 0) {
        // Get the trackable
        const QCAR::Trackable* trackable = state.getActiveTrackable(0);
        currentModelViewMatrix = QCAR::Tool::convertPose2GLMatrix(trackable->getPose());

        for (int j=0; j<objects3D.count; j++) {
            QCAR::Matrix44F modelViewMatrix = currentModelViewMatrix;
            QCAR::Tool::convertPose2GLMatrix(trackable->getPose());
            Object3D *obj3D = [objects3D objectAtIndex:j];

            // Render using the appropriate version of OpenGL
            if (QCAR::GL_11 & qUtils.QCARFlags) {
                // Load the projection matrix
                glMatrixMode(GL_PROJECTION);
                glLoadMatrixf(qUtils.projectionMatrix.data);

                // Load the model-view matrix
                glMatrixMode(GL_MODELVIEW);
                glLoadMatrixf(modelViewMatrix.data);

                glScalef(kObjectScale, kObjectScale, kObjectScale);

                // Draw object
                glTranslatef(0, 0, 0.00001*j);
                glTranslatef(obj3D.dx * 0.01, obj3D.dy * -0.01, 0.0);
                glRotatef(-obj3D.rotation, 0, 0, 1);
                glScalef(obj3D.scale, obj3D.scale, 1.0);

                glBindTexture(GL_TEXTURE_2D, obj3D.texture.textureID);
                glTexCoordPointer(2, GL_FLOAT, 0, (const GLvoid*)obj3D.texCoords);
                glVertexPointer(3, GL_FLOAT, 0, (const GLvoid*)obj3D.vertices);
                glNormalPointer(GL_FLOAT, 0, (const GLvoid*)obj3D.normals);
                glDrawElements(GL_TRIANGLES, obj3D.numIndices, GL_UNSIGNED_SHORT, (const
GLvoid*)obj3D.indices);
            }
        }
    }

    // Disable certain GL states
    ...

    QCAR::Renderer::getInstance().end();
    [self presentFramebuffer];
}
```

Initially, a framebuffer is prepared and the QCAR renderer's begin() function has to be called to probably setup a QCAR state. Then, using this state, the number of active Trackables can be queried. You can loop over the range if you want to support multiple Trackables. In our case, we only support a single Trackable. To get the active trackable call state.getActiveTrackable(i), where i presents a valid active Trackable index.
Using trackable->getPose() the ModelView matrix is acquired and transformed into OpenGL format. This can then be used to perform OpenGL ES rendering like setting up the ModelView and Projection matrizes, etc. The code for OpenGL ES 2.0 has been omitted here. Finally any enabled OpenGL states are disabled again, the renderer's end() method has to be called and the framebuffer is presented.