### **Computer Science 4ZP6 - Final Report**

### Spineducation

## Maya Ramamurthy, Katrine Rachitsky, Manaar Hyder, Randa Mohsen

## Christopher Anand, Colby Oitment

# **Overview**

Spineducation is an educational application geared towards preparing students in varying stages of becoming physicians by providing them with the theory and practice they need. Our project aims to provide users with replications of real life clinical scenarios and multiple-choice questions that will help prepare students for their medical exams. Additionally, our project allows medical students to practice performing procedures inside an augmented reality environment. When it comes to hands-on experience, medical students have very few opportunities to practice their spinal surgery skills due to its intricate nature; even minor mistakes can cause paralysis at best, and death at worst. As such, the user of a medical application like ours would be provided both, a virtual practical experience through an augmented reality game which tests surgical skill, and a study environment, teaching theoretical knowledge through cases and multiple-choice questions.

With the help of our application, medical students and residents do not need to potentially endanger the life of a patient in order to go through their necessary training; Instead of making decisions about a real patient’s treatment plan, which has very high stakes and could ultimately be fatal, our application provides users the opportunity to go through a hypothetical medical case step by step and learn how a real doctor should handle a variety of situations. Similarly, instead of endangering the life of a patient by operating on them with little to no training, a resident could use our application to simulate a surgical environment where a spine is superimposed into the user’s environment. In this way a user is able to inspect the different anatomical features of the spine, analyze each vertebra to understand the structure of the spine as a whole, and most importantly, actually perform a simple surgery and receive feedback on skill level and areas of improvement.

The augmented reality side of the application can be broken down into a few separate components, which together form the augmented reality surgery game: 3D Models, selection of start points, trajectory, screw placement, and verification of surgical skill. The rest of the application can be broken down into 3 main sections: Medical cases, images, and multiple choice questions, the latter two of which fit into the former.

## **3D Models**

An existing 3D model of the spine was modified in Blender to customize it for our application. This meant reformatting the 3D image (it initially came with large spaces between each vertebra, due to the file type’s lack of compatibility with Blender, which was fixed by removing the formatting associated with the initial file type), changing its rotation, size, texture, and material. Next, 3D cylinders were painstakingly created with the help of a surgical resident and customized to fit within the vertebra of the spine; these 3D cylinders represent the acceptable area for a surgical screw to the be placed, also known as an acceptable area for a ‘start point’. These help the application determine whether or not the screw placement of the user is correct. Additionally, a 3D model of the screw is placed into the spine once the user has selected a start point and trajectory, it’s rotation and trajectory adjusted to the user’s specification.

## **Selection of Start Points**

Upon launching the AR portion of the app, the user is prompted to place the spine on the screen. Upon clicking a point on the screen, the method touchesBegan() is called for the first time. This method will recognize location which was touched on the screen by reading it in from the ARkit UITouch event. After this, the spine is placed relative to the user’s camera position to show up in an optimal position.

When the spine is superimposed into the user’s environment, a 2D image of a bullseye is attached to the user’s “point of view” in the augmented reality scene view so that it moves with the user. This 2D image is used to help the user close in on the start point that they would like to select on the spine. Upon the selection of a start point, the same touchesBegan() method is called, however this time, through the use of Boolean variables, the code moves on to the portion related to the start point selection. Through the use of a “hit test” function, the code checks where the user touched in relation to checking if a node was hit, or if the user touched somewhere other than the spine. The code, if a node was hit, will then preform a world transformation upon the hit position to convert the Real-World coordinates to SceneView coordinates and set that position to be the start point.

Once a start point has been selected a small white ball will appear, to indicate the placement of the selected start point. A line will appear from the start point to the user’s point of view in the horizontal plane of the user’s environment to allow for trajectory selection. Note that the start point and the trajectory must both lie in the cylinder (which is hidden from the user when the spine model is shown in the application) in order to constitute an acceptable placement selection for the screw.

## **Trajectory**

Once trajectory has been selected, the user must use the line starting at the start point and ending at the user’s point of view, done by dynamically realigns itself as the user moves about in the augmented reality environment. The code to generate this is done by generating a new line object upon movement of the camera, and deleting the old line object, so that it appears as if the line is following the camera position. This will determine a trajectory with which to enter the vertebra. The user moves the camera to move the line according to the camera angle and select a trajectory with which they mean to place the screw into the pedicle start point on the spine.

## **Screw Placement**

If the user has selected a start point, and then a trajectory, the screw model can now be displayed to the user and placed in the spine according to the specifications of the user (in the start point position, at the rotation of the trajectory). The screw uses the built-in feature to obtain the camera angle and rotation of the user in order to determine the rotation the user means to place the screw. Through the use of inverting the y and z components of both the camera’s point of view rotation we are able to configure the orientation such that the screw is placed in the exact orientation and rotation that the user has specified through the movement of the device, which the trajectory line helps the user visualize. The code for determining the exact orientation and rotation can be seen below:

let camera = self.sceneView.pointOfView!

screwModel = screw.rootNode.childNode( withName: nodeName, recursively: true)

screw.rootNode.position = position;

screw.rootNode.rotation = SCNVector4(camera.rotation.x, -camera.rotation.y, -

camera.rotation.z, camera.rotation.w);

screw.rootNode.orientation = SCNVector4(camera.orientation.x, -camera.orientation.y, -

camera.orientation.z, camera.orientation.w)

sceneView.scene.rootNode.addChildNode(screw.rootNode)

This is done in front of the user, so that the user can view a visual representation of the important surgical step they have completed. The user has now placed a screw into a vertebra of the spine, and all that is left is for the application to verify the placement and provide feedback to the user.

## **Verification**

If the start point selected and the trajectory selected are both calculated to be ‘within’ the cylinder objects placed within the spine model, the user is notified that they have properly placed the surgical screw, and are allowed to try again on a different vertebra. The precise way that it is determined is broken down into two main components. For one, a hit test is preformed between the user’s start and end points (as selected in an earlier stage) to determine if it intersects with the cylinder node in question. The second is to determine if the cylinder node was selected by the user when choosing the start point. If either of these conditions were met, it means that the user was successful in regard to placing the screw object. Otherwise, the user is notified that their placement was faulty, and they may try again on the same vertebra.

## **Hit Tests**

The AR component refers to Hit Tests quite a bit, as it was the main way of determining if any two objects, be it user touch and node, node and node, ect, collided. This is done by taking the 2D (x,y) coordinates of the user touch and creating a line of sorts through the camera origin to the 2D point which has a sort of 3D position on the plane. And if any plane or node intersects with the line, then we know that something was at the location selected. From this result, we can obtain the world 3D coordinate which the user selected in the 2D plane

## **Medical Cases**

Once the user selects the cases option, they are shown a list of cases to choose from. Each case leads to a page that displays the patient’s general history. There will also be information from their physical, as well as links to the pre-op CT images. After the user has all of the information, they can proceed to the relevant multiple-choice questions, which will quiz them on the information they have just obtained.

## **Images**

Thousands upon thousands of surgical cases are worked upon by surgeons all year round. What make these cases individually unique, are their 2D and 3D images. 100+ images of real spine deformities are often saved after each patient undergoes CT or MRI scans. These images are extremely valuable, as they cannot be created from scratch; even graphical creativity cannot accurately design a spinal deformity that’s as realistic as these images. Our app provides the platform for students to access these images for educational purposes. Our team of medical surgeons work carefully to anonymize the images and incorporate them into the medical cases mentioned above.

## **Multiple Choice Questions**

The multiple-choice questions quiz users on their general knowledge of spines. Depending on whether the user has selected the speed round version or the cases version, the questions will be either random or pertaining to the case in progress. Users will see a question, highlighted at the very top in blue, and 4 options, each highlighted in grey. If the user selects the correct answer, the option will turn green. If the user selects the wrong answer, the selected option will turn red, and the correct answer will turn green. Thus, the user will be able to immediately learn from their mistakes.

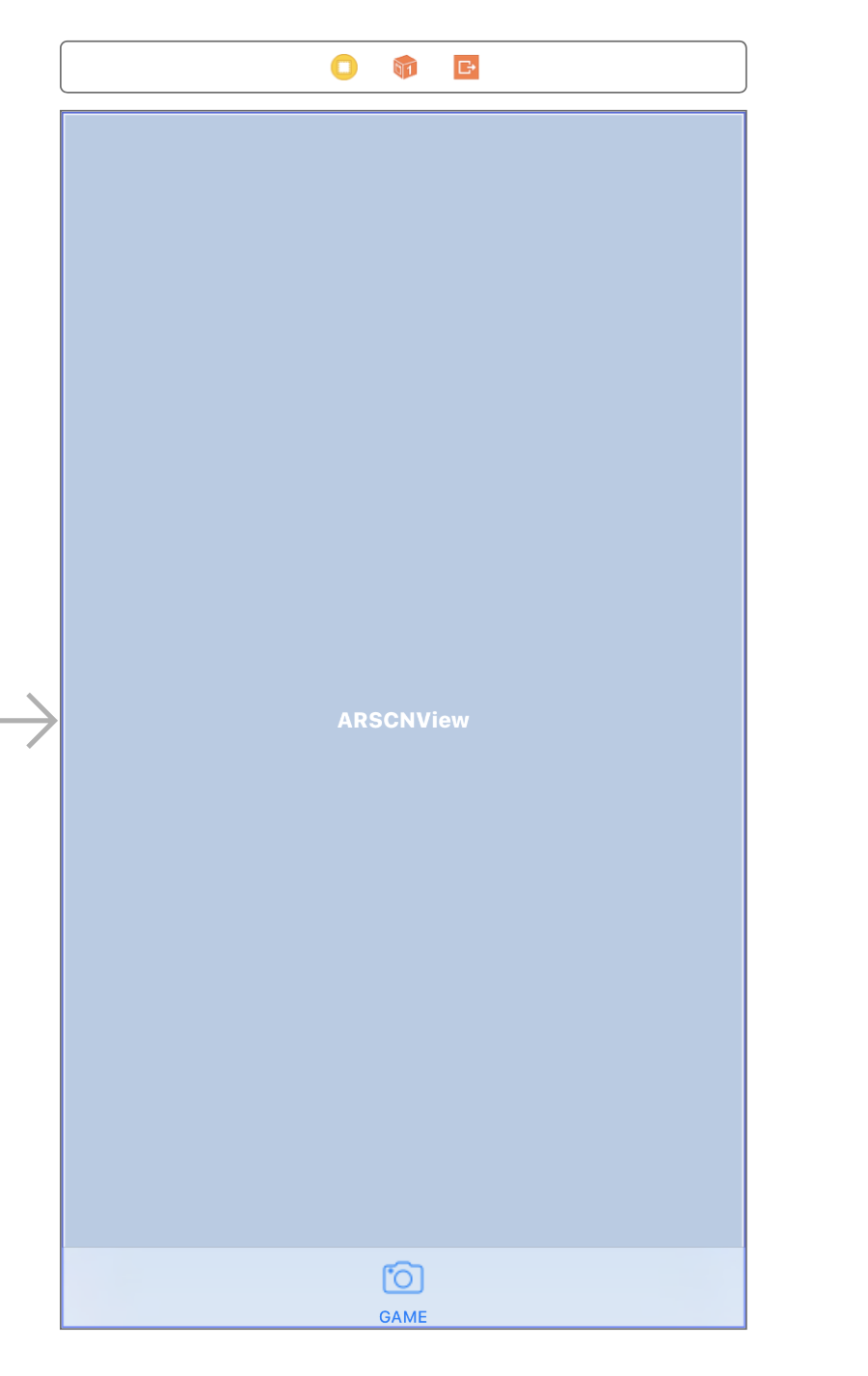
The app will also keep track of the user’s performance with the questions. There will be a profile page to display the different categories and how well the user performed in each of them, as well. This will be helpful for the user to know which areas require improvement, and thus practice those areas until perfect, ultimately having a higher chance of becoming a better doctor and helping more patients.

# **Augmented Reality (AR) Component**

## *AR Code Base:*

## **Storyboard Component:**

The Storyboard for this component is fairly simple, with just an *ARSCNView* component added into the screen to allow for the AR function to work. This part is tied into the “GAME” component of the menu options on the bottom bar.



## **ViewController.swift:**

There are several methods in this class which allow the AR component to work:

override func viewDidLoad()

This is the first function that is called upon in ViewController.swift. This function is used to set up the key components for AR, such as initializing the Scene, and displaying the first instruction for tapping the screen to show the spine to the user.

func showTarget()

This function is used to create and place the 3D bullseye object on the scene

at the camera position

func showUserInstruction ()

This function is created so that it is easier to place text on the plane by sending

in the string of what is to be placed and an offset. In turn, this function will

create the text nodes and set the material, the place it relative to the camera

func make2dNode()

This function will create a 3D node out of a 2D image. Given the bullseye image

is a 2D image, it will allow it to show up as a 3D object.

func doHitTestOnExistingPlanes()

Does a hit test on the horizontal plane to check if clicked position was on the

plane.

func positionFromTransform()

Will Transform the hit test node real world position to AR world position.

func nodeWithPosition()

Gets the position from the touched point of the select start point, and will

create a sphere as an anchor for the line to use as the start point position. It

is also where the screw will be placed.

override func touchesBegan()

This function is where the majority of the AR Surgery occurs, as it is where the app detects the user’s taps onto the screen and responds accordingly, depending on the value of a variety of variables. There are a few global variables initialized in ViewController.swift, mainly used to indicate the state in which the program is currently in.

if (!targetExists)

If the target does not exist already, it will be added to the scene. Moreover, it will be added as a child node of the user perspective, so that it remains in the middle of the screen, regardless of where the user moves in 3D space. The target not existing also indicates that the spine has not yet been displayed to the user (this is likely the first screen tap), so the spine will be added to the scene as a child node of the scene, so that it will behave as a real 3D object in 3D space. Additionally, the user will be given a few seconds to reposition the spine by tapping the screen so that the spine is recreated in any direction the user points the camera of their device.

(Calls on the createSpine() and the showTarget()functions)

if (targetExists && !targetLocked)

Values of true and false for these variables, respectively, indicate that the above target has been created, however the start point for the surgery has not yet been set. This will call upon the SCNHitTestOption to determine if the user has selected a valid part of the spine. If not, the user will be able to continue tapping until they have. Once they do select a valid component of the spine, this point will be ‘locked in’ and saved for use in determining trajectory. Once they click it will also allow them to see the line for the trajectory angle to select a trajectory.

if (targetLocked && !trajectoryExists)

This will check if the trajectory exists and if not, it will lock in the angle of the user at that moment. It calls the createScrew() method, to place the screw. Then will do a hit test to see if the screw intersects with any ‘cylinders’ to check if the user hit a pedicle, or a random point. The user will then be informed if they did the surgery correctly, and will be allowed to try again.

func showTarget()

This function will make an image of a 2D target (make2dNode) into a node and

attach it to sceneView.pointOfView?.addChildNode(bullseyeNode) so that it appears

on the camera position.

  func showUserInstruction ()

This function allows easy drawing of instructions on the screen by sending it an

offset value for proper positioning and the string, holding any desired instruction.

  func make2dNode()

Creates a 3D node object from a 2D image

  override func viewWillAppear()

Creates a session configuration for AR.

override func viewWillDisappear()

Pauses the view’s session (ie if switch screens can reload current progress)

func createSpine(position : SCNVector3)

        Creates the spine node at the given position, and attaches it to the scene.

func createSpine(position : SCNVector3)

Creates the screw at the start position selected. Will get the camera angle and

rotation from the camera and place the screw in at that angle and rotation.

func renderer(\_renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor)

Called when a SceneKit node corresponding to a new AR anchor has been added to

the scene. Will draw the plane.

func renderer(\_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval)

Will allow for the trajectory line to be redrawn each time the camera moves so

it appears as if the line is following the camera.

func renderer(\_renderer:SCNSceneRenderer, didUpdate node:SCNNode, for anchor: ARAnchor)

Called when a SceneKit node's properties have been updated to match the current

state of its corresponding anchor. Will redraw the plane

func normalizeVector()

Helper function meant to normalize the vector for the cylinder line.

extension SCNGeometry

class func lineFrom()

Will create a line from the start point and store the start and end

variables

extension SCNNode

func cylinderLine

Function meant to draw a trajectory line given the start position and

end position which is the selected point to the camera. Does the

calculations necessary to draw this line.

    override func didReceiveMemoryWarning()

      Release any cached data, images, etc that aren't in use.

    func session()

        Present an error message to the user if failed to load session

    func sessionWasInterrupted()

        Inform the user that the session has been interrupted, for example, by

presenting an overlay

func sessionInterruptionEnded()

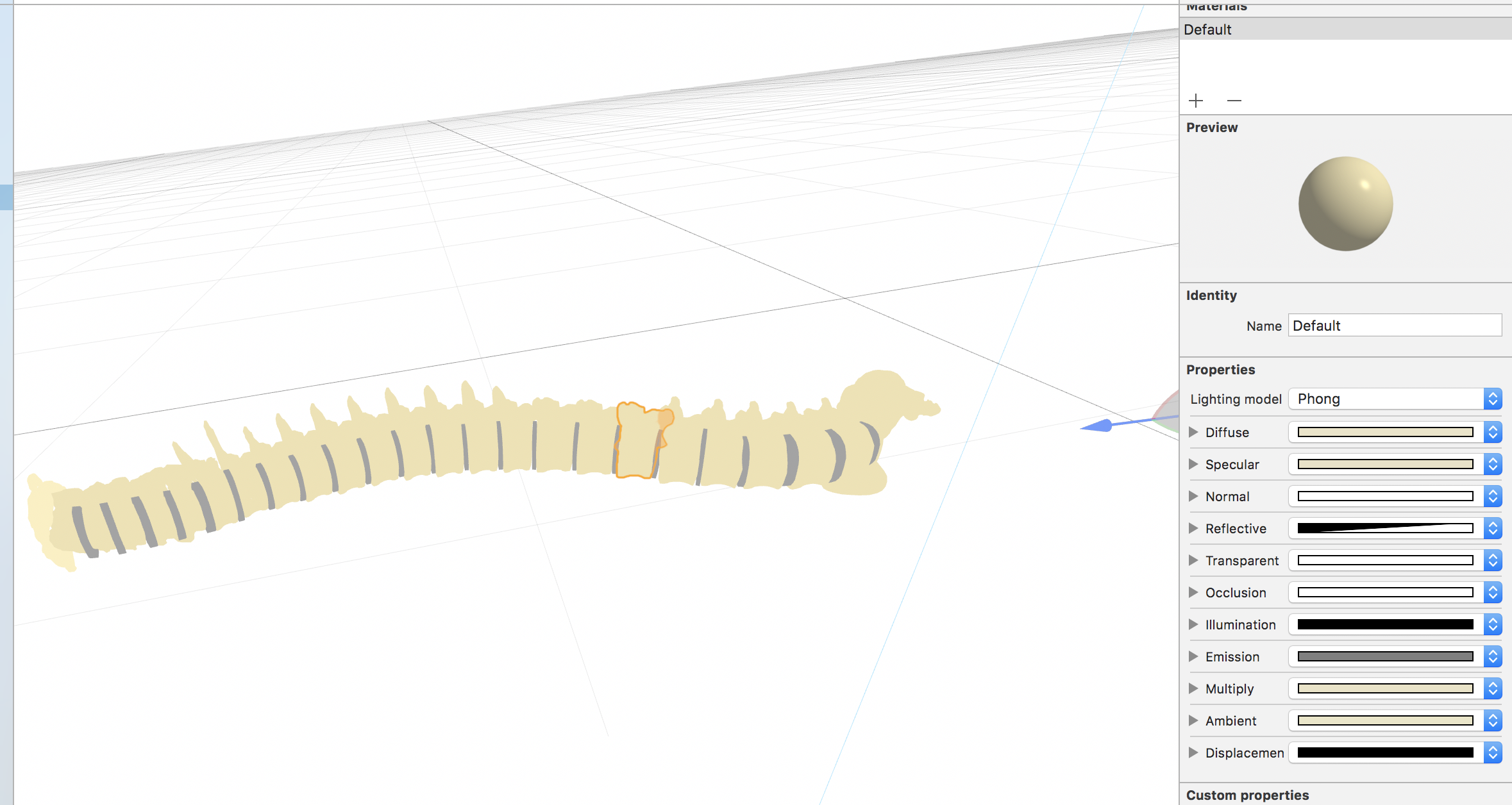
        Reset tracking and/or remove existing anchors if consistent tracking is

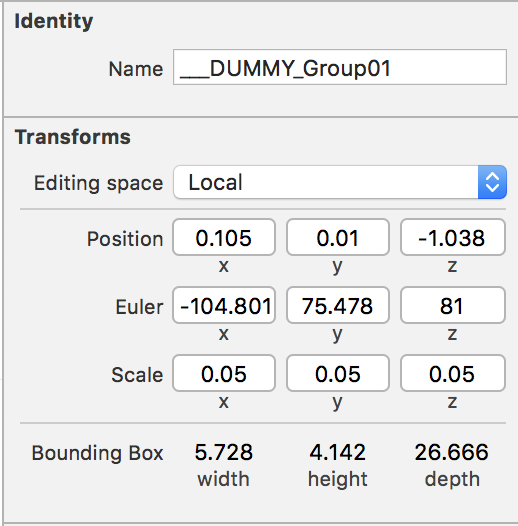
required

# *Spine Models:*

The current spine model is 4Cylinders.dae. This file has been modified in Blender to add in cylinders for the surgery components. These cylinders indicate acceptable pedicle start points as well as encompassing the entirety of the acceptable trajectory for a screw to enter the spine (if the start point or trajectory lie outside of a cylinder, they can be deemed inaccurate and the user has failed to properly execute the procedure). Additionally, it has the following XCode properties attached to it in order to add textures, materials to it, and colouring. The cylinders are marked as “hidden” so that, while they exist on the spine model, the user cannot see them, which adds difficulty.

Please also note in the second screenshot, the numbers corresponding to the position, euler and scale rows are not to be changed, this can ruin the proper positioning of the spine.



**

# *Getting Started with AR (Useful Resources)*

Learning How to set up AR Environment

* <https://lifehacker.com/how-to-get-started-using-apple-s-arkit-augmented-realit-1797690723>
* <https://blog.pusher.com/building-an-ar-app-with-arkit-and-scenekit/>
* <https://developer.apple.com/documentation/arkit/handling_3d_interaction_and_ui_controls_in_augmented_reality>
* <https://blog.markdaws.net/arkit-by-example-part-2-plane-detection-visualization-10f05876d53>
* <https://gorillalogic.com/blog/arkit-developer-tutorial-how-to-build-a-shoe-measuring-app/>

## Adding Objects and Materials/Textures in Blender:

* <https://www.wikihow.com/Apply-a-Material-or-Texture-in-Blender>
* <https://blender.stackexchange.com/questions/8697/how-do-i-put-an-image-texture-on-a-material-in-cycles>
* <https://www.youtube.com/watch?v=JFdVRdD9VSM>

## Open Source Spine Models

* <https://grabcad.com/library/human-spine-1>
* <https://www.thingiverse.com/thing:31845/#comments>
* [https://sketchfab.com/models/1ce94666922f48c9b68cd0e196b74a0a#](https://sketchfab.com/models/1ce94666922f48c9b68cd0e196b74a0a)
* <https://grabcad.com/library/model-pedicle-screw-solidworks-iges>

## *AR Related Errors:*

### **Help! The Spine Is Flying Away:**

Sorry, this cannot be helped. This error occurs due to the fact that AR works by configuring itself to the camera’s real-world surroundings, and, as such, places the object (ie the Spine) in real world configuration. This is common when the user is moving the camera too jerkily, or switching the surroundings quickly, as this will cause the spine to fly away as the program is attempting to read the surroundings and place an object in an unstable 3D space.

**What you can do:** Hold the camera steady and ensure that your environment is adequately lit (poor lighting can result in a poor read of the environment, the phone camera cannot determine the planes of the 3D space when the surroundings are indiscernible), and the AR will re-configure based on the surroundings and stabilize. If this does not work, then try relaunching the app. It is very rare that the app is unable to reconfigure once the surroundings have been stabilized and the environment has enough lighting. Also try using a different view on the camera (ie turn a different direction) as it is possible the direction you were trying was too confusing for the AR. You should see a plane focused in on the ground, and the console should display that the camera state is ‘normal’, and once it has done this, AR has been properly configured.

### **My Screw Is Going in Wrong, How Do I Fix This:**

While the trajectory line makes for a nice visual, the entirety of the calculations relating to the trajectory is done based on the camera angle. This is due to the fact that it is extremely difficult in AR to be able to calculate the angle based on just the line itself.

**What you can do:** If you are finding that the screw is not going in how you wanted, make sure you are rotating the device at the exact angle you want it to go in at, and the screw will be placed in accordingly.

# **iOS User Interface Component**

## Components

Launch Screen

The first screen users will see when opening the app. It is made up of the following components

* A View Controller - This will act as the container for each screen
  1. At the very bottom of the panel on the right, there is a search bar. Use that, and then drag and drop the View Controller onto the main screen
* An Image viewer - this is for the background photo of the skeletal spine
  1. From the bottom portion of the right-side panel, drag and drop an image viewer into the view controller. Stretch it to cover the whole VC
  2. Ensure that the image you wish to use is within the project repository
  3. Select the image viewer, and then in the right-hand panel, go to the attribute inspector tab
  4. Under the Image view section, choose the image that should go there
  5. Set the view to “scale to fill”
  6. Make sure it’s opaque, not hidden, and clipped to bounds
  7. In the bottom panel, click the add new constraints button, make sure there is 0 spacing to nearest neighbour on each side

Title Screen

This page has the same look and feel as the launch screen, except it has a title label and a log in button

* A Label - this is for the project title
  1. From the bottom portion of the right-side panel, drag and drop an image viewer into the view controller
  2. Select the label, and then in the right-hand panel, go to the attribute inspector tab
  3. Select the plain text type and then type the title right under that. Make sure it is black, centred, enabled and highlighted
  4. Set a white shadow with an offset of 3
  5. Set the background to be a light text colour. Ensure that it is not opaque in this case.
* A button - this is for the project title
  1. From the bottom portion of the right-side panel, drag and drop a button into the view controller
  2. Select the button, and then in the right-hand panel, go to the attribute inspector tab and ensure the settings look like this:

* 1. Add another VC. Hold down the control key. Click the button and drag the mouse over to the next VC. Select ‘show modally’. Now when the button is clicked, it will lead to that VC (page), which is the Study tab, allowing the user to select the “medical cases” mode or the “speed round” mode.
* Stack view – Horizontal and vertical stacks help with keeping component aligned relative to other objects, so the layout looks the same across all platforms
  1. Drag and drop the stacks from the object library in the bottom right
  2. Add constraints, the same way they were added for the image in the launch screen
  3. The objects that should be within the stack can now be dragged over the stack in the document outline until the blue line gets shorter and the object is now indented under the stack in the document outline
  4. The object’s constraints may need to be adjusted

Study Tab

This page allows users to choose whether they would like to go through cases, or be given stand-alone multiple-choice questions

* Buttons – Added same way as described in the title screen section. Should be called “medical cases” and “speed round”. Each button will lead to a Spinner screen where the user can be more specific about the types of questions/cases wanted.
* Stack view– Horizontal and vertical stacks help with keeping component aligned relative to other objects, so the layout looks the same across all platforms
  1. Drag and drop the stacks from the object library in the bottom right and set it up, same as it was done for the title screen
  2. Ensure the two buttons are within the stack view

Category Screen (both)

There will be two identical screens like this. One for the med cases, and one for the speed round. It allows users to choose a surgical category and anatomical region from two spinners. The cases/ questions will then be provided accordingly

* Button – Added same way as described in the title screen section. Should be called “Go”. Used to move to the next page once selections have been made.
  1. If it is the spinner for the Speed round, the next page will be the questions page.
  2. If it is the spinner for the medical cases, the next page will be the category info page
* Label – There are two labels on this page provide instructions on what to do with each spinner. Set them up like the label from the title screen, but make the background blue
* Stack view – add in a stack view to contain the labels, spinners, and button

##### Picker view

* 1. Drag and drop the Picker View from the object library in the bottom right. Position it and place constraints appropriately.
  2. Create a class of type UIViewController and set it to be the custom class of this page.
  3. Open the assistant editor to help you make a reference of the picker view in the code of the class.

@IBOutlet weak var SurgicalCategoryPicker: UIPickerView!

* 1. Create an array to hold the values you need to appear in the picker view

var SurgicalCategoryOptions = ["All","Trauma", "Tumour", "Deformity", "Infection", "Degenerative", "Pediatric", "Adult", "Random"]

* 1. Define function to tell the app the number of rows this picker view will hold

func pickerView(\_ pickerView: UIPickerView, numberOfRowsInComponent

component: Int) -> Int

In this case, the return value will be the size of the array created in step 4.

* 1. Define function to tell the app the number of columns this picker view will hold

func numberOfComponents(in pickerView: UIPickerView) -> Int

* 1. Define function to tell the app the data to be returned when the picker view is used by the user. In our case, the index of the value in the picker view that they chose will be used to search the array created in step 4.

Questions page

This page will have a question at the very top, and then 4 choices for the user to select the correct answer

* Button – Added same way as described in the Study tab section. Should be called “Next MCQ”. Used to move to the next question once selections have been made
* Labels - There are 4-5 labels on this page. Set them up like the labels from the spinner screen
  1. The first one provides the question and has a blue highlight.
  2. The rest of them are the answer options what to do with each spinner. Set them up like the label from the title screen, but make the background blue
* Stack view – add in 2 stack views to contain the labels
  1. The blue highlighted question label should have 0 spacing between the left and right edges of the screen
  2. The grey highlighted answer options should have some spacing, so they don’t quite reach the side edges of the screen.
* Connection to the database should be documented at a later time.

## Category Info Page

* Stack view – add in 2 stack views to contain the labels
  1. The blue highlighted “A random case has been chosen for you” label should have 0 spacing between the left and right edges of the screen
  2. All the other objects should have some spacing, so they don’t quite reach the side edges of the screen.
* Button – Added same way as described in the study tab section, but also with a blue background. Should be called “BEGIN”.
* The info for this case has been hard coded so far.

## Case Info Page

* Labels - There are 4-5 labels on this page. Set them up like the labels from the spinner screen
  1. Three of them are subheadings, highlighted in blue
  2. Two of them are just info, no background.
* Stack view – add in 2 stack views to contain the labels
  1. The 3 blue highlighted subheading label should have 0 spacing between the left and right edges of the screen
  2. The 2 blue info labels should have some spacing, so they don’t quite reach the side edges of the screen.
* Button
  1. One button should be called “Continue” Which allows the user to go to the next page, whether it be the MCQ questions of the next prompt. The questions on the page now will only be questions relevant to the case.

## Navigation Controller

This is placed ahead of the study section of the app in order to allow the user to go back to their previous page at any time. The way it works: for anything you want to lead to the study page, you use a segue from there to the navigation controller. The navigation controller will always segue to the Study Tab.

# **Common XCode Related Errors**

## *Error Message:* “Code Signing Error”

### Details:

Code Signing Error: No account for team "NWVTD7H53D". Add a new account in the Accounts preference pane or verify that your accounts have valid credentials.

Code Signing Error: No profiles for 'com.<name>.Spineducation' were found:  Xcode couldn't find any iOS App Development provisioning profiles matching 'com.<name>.Spineducation'.

Code Signing Error: Code signing is required for product type 'Application' in SDK 'iOS 11.2'

### Cause:

This error occurs due to the fact that when the Git Project was pulled from Git, the name of the team was left from the last user’s details.

### Solution:

This can be fixed in the *Spineducation.xcodeproj* file by ensuring that the proper values are in each field.



## *Error Message: “Command /usr/bin/codesign failed with exit code 1”*

Details:

CodeSign /Users/mayaramamurthy/Library/Developer/Xcode/DerivedData/Spineducation-binpzylunoyxhpfekgermhvvuupp/Build/Products/Debug-iphoneos/Spineducation.app

    cd "/Users/mayaramamurthy/Documents/School/University/Fourth Year/CS 4ZP6/Git project/Spineducation/Spineducation"

    export CODESIGN\_ALLOCATE=/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/codesign\_allocate

    export PATH="/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/usr/bin:/Applications/Xcode.app/Contents/Developer/usr/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin"

Signing Identity:     "iPhone Developer: mayaramamurthy@gmail.com (3JUPN965XP)"

Provisioning Profile: "iOS Team Provisioning Profile: com.csgirls.Spineducation"

                      (f0e35f3d-b96a-49f9-a6a3-084128847d79)

    /usr/bin/codesign --force --sign 489B4469BBB64D3611EA48051C3E878AF6DB9768 --entitlements /Users/mayaramamurthy/Library/Developer/Xcode/DerivedData/Spineducation-binpzylunoyxhpfekgermhvvuupp/Build/Intermediates.noindex/Spineducation.build/Debug-iphoneos/Spineducation.build/Spineducation.app.xcent --timestamp=none /Users/mayaramamurthy/Library/Developer/Xcode/DerivedData/Spineducation-binpzylunoyxhpfekgermhvvuupp/Build/Products/Debug-iphoneos/Spineducation.app

/Users/mayaramamurthy/Library/Developer/Xcode/DerivedData/Spineducation-binpzylunoyxhpfekgermhvvuupp/Build/Products/Debug-iphoneos/Spineducation.app: resource fork, Finder information, or similar detritus not allowed

Command /usr/bin/codesign failed with exit code 1

Cause:

This error occurs because of cached information from pulling from Git

Solution: Run the following line in Terminal:

*xattr –rc ~/Library/Developer/Xcode/DerivedData*