ДОДАТОК А

Фрагменти коду програми

```
protocol CircularBufferElementType {
    static var zero: Self { get }
    static func + (_ lhs: Self, _ rhs: Self) -> Self
    static func / (_ lhs: Self, scalar: Float) -> Self
}

class CircularBuffer<T: CircularBufferElementType> {
    private let size: Int
    private let dequeueCount: Int
    private var array: [T] = []

    var average: T { array.reduce(.zero, +) / Float(array.count) }
    var isFull: Bool { size == array.count }

    init(size: Int = 30, dequeueCount: Int = 5) {
        self.size = size
        self.dequeueCount = dequeueCount
    }

    static func + (_ buffer: CircularBuffer, _ value: T) {
        if buffer.array.count > buffer.size {
            buffer.array.removeFirst(buffer.dequeueCount)
        }

        buffer.array.append(value)
    }
}

extension SCNVector3: CircularBufferElementType { }

class SightInteractionHandler {
    private typealias InteractiveView = UIView & SightInteractable

    private weak var view: UIView!

    init(view: UIView) {
        self.view = view
    }

    private var lastFocusedView: InteractiveView?
    func processSightPoint(_ point: CGPoint) {
        guard let target = view.hitTest(point, with: nil) as?
InteractiveView else {
            defocusIfNeeded()
            return
        }

        if target !== lastFocusedView {
            defocusIfNeeded()
        }

        let innerPoint = target.convert(point, from: view)
```

```
        target.handleSightFocus(at: innerPoint)
        lastFocusedView = target
    }

    private lazy var eventHandler: (FaceExpressionEvent) -> Void =
throttle(delay: 1) { [weak self] event in
        self?.commit(event)
    }

    func handleFaceExpressionEvent(_ event: FaceExpressionEvent) {
        eventHandler(event)
    }

    private func commit(_ event: FaceExpressionEvent) {
        guard event == .smile else { return }
        guard let focusedView = lastFocusedView else { return }

        if let control = focusedView as? UIControl {
            control.sendActions(for: .touchUpInside)
        }
    }

    private func defocusIfNeeded() {
        lastFocusedView?.handleSightDefocus()
        lastFocusedView = nil
    }
}

extension SCNVector3: Hashable {
    public func hash(into hasher: inout Hasher) {
        hasher.combine("\(x),\(y),\(z)")
    }
}

extension SCNVector3 {
    func cgPoint(adjustScaleFactor: Bool = false) -> CGPoint {
        if adjustScaleFactor {
            let scale = UIScreen.main.scale
            return CGPoint(x: CGFloat(x) / scale, y: CGFloat(y) / scale)
        }

        return CGPoint(x: CGFloat(x), y: CGFloat(y))
    }
}

extension SCNVector3: Equatable {
    static var zero: SCNVector3 {
        return SCNVector3Zero
    }

    var length: Float {
        return sqrtf(x * x + y * y + z * z)
    }

    var normalized: SCNVector3 {
        return self / length
    }
```

```
    func distance(to vector: SCNVector3) -> Float {
        return (self - vector).length
    }

    init(m: matrix_float4x4) {
        self.init(m.columns.3.x, m.columns.3.y, m.columns.3.z)
    }
}

public func == (lhs: SCNVector3, rhs: SCNVector3) -> Bool {
    return lhs.x == rhs.x && lhs.y == rhs.y && lhs.z == rhs.z
}

public func * (vector: SCNVector3, scalar: Float) -> SCNVector3 {
    return SCNVector3(vector.x * scalar, vector.y * scalar, vector.z *
scalar)
}

public func *= (vector: inout  SCNVector3, scalar: Float) {
    vector = vector * scalar
}

public func / (vector: SCNVector3, scalar: Float) -> SCNVector3 {
    return SCNVector3(vector.x / scalar, vector.y / scalar, vector.z /
scalar)
}

public func /= (vector: inout  SCNVector3, scalar: Float) {
    vector = vector / scalar
}

public func + (l: SCNVector3, r: SCNVector3) -> SCNVector3 {
    return SCNVector3(l.x + r.x, l.y + r.y, l.z + r.z)
}

public func += (l: inout SCNVector3, r: SCNVector3) {
    l = l + r
}

public func - (l: SCNVector3, r: SCNVector3) -> SCNVector3 {
    return SCNVector3(l.x - r.x, l.y - r.y, l.z - r.z)
}

public func -= (l: inout  SCNVector3, r: SCNVector3) {
    l = l - r
}

public func * (l: SCNVector3, r: SCNVector3) -> SCNVector3 {
    return SCNVector3(l.x * r.x, l.y * r.y, l.z * r.z)
}

public func *= (l: inout  SCNVector3, r: SCNVector3) {
    l = l * r
}

public func / (l: SCNVector3, r: SCNVector3) -> SCNVector3 {
    return SCNVector3(l.x / r.x, l.y / r.y, l.z / r.z)
}
```

```
public func /= (l: inout  SCNVector3, r: SCNVector3) {
    l = l / r
}

public func min(_ l: SCNVector3, _ r: SCNVector3) -> SCNVector3 {
    let ld3 = SIMD3<Double>(l)
    let rd3 = SIMD3<Double>(r)

    return SCNVector3(min(ld3, rd3))
}

public func max(_ l: SCNVector3, _ r: SCNVector3) -> SCNVector3 {
    let ld3 = SIMD3<Double>(l)
    let rd3 = SIMD3<Double>(r)

    return SCNVector3(max(ld3, rd3))
}

enum FaceExpressionEvent: Hashable {
    case smile
    case blinkLeft
    case blinkRight
}

class FaceTracker {
    weak var view: ARSCNView!

    var onNextPoint: ((CGPoint) -> Void)?
    var onFaceEvent: ((FaceExpressionEvent) -> Void)?

    var pointOfView: SCNNode? {
        return view?.pointOfView
    }

    var isReady: Bool {
        return currentFaceAnchor != nil
    }

    var currentFaceAnchor: ARFaceAnchor?
    var currentFaceNode: SCNNode?

    var faceGeometry: ARSCNFaceGeometry? {
        return currentFaceNode?.geometry as? ARSCNFaceGeometry
    }

    lazy var screenNode: SCNNode = {
        let screenGeometry = SCNPlane(width: 0.2, height: 0.2)
        let node = SCNNode(geometry: screenGeometry)

        node.geometry?.firstMaterial?.isDoubleSided = true
        node.name = "Phone plane"
        node.position = SCNVector3(0, 0, -0.03)
        node.opacity = 0

        return node
    }()
```

```
    lazy var deviceNode: SCNNode = {
        let node = SCNNode(geometry: nil)
        node.addChildNode(screenNode)
        node.name = "Device node"
        return node
    }()

    lazy var focusNode: SCNNode = {
       let node = SCNNode(geometry: SCNSphere(radius: 0.001))
        node.geometry?.firstMaterial?.diffuse.contents = UIColor.green
        return node
    }()

    var head = Head()
    var gazePoints: [CGPoint] = []

    func configure(with view: ARSCNView) {
        self.view = view

        view.scene.rootNode.addChildNode(deviceNode)
        view.scene.rootNode.addChildNode(head)
        view.scene.rootNode.addChildNode(focusNode)
    }

    func updateNode(_ node: SCNNode, for anchor: ARFaceAnchor) {
        self.currentFaceNode = node
        self.currentFaceAnchor = anchor

        guard let geometry = faceGeometry else {
            return
        }

        geometry.update(from: anchor.geometry)
        head.update(withFaceAnchor: anchor)
        sendFaceEvents(for: anchor)
    }

    func update(with rendered: SCNSceneRenderer) {
        guard isReady else { return }
        guard let pointOfView = pointOfView else { return }
        deviceNode.transform = pointOfView.transform

        guard let lEyeHitTest = hitTest(node: deviceNode, gaze:
leftEyeGaze()).first else { return }
        guard let rEyeHitTest = hitTest(node: deviceNode, gaze:
rightEyeGaze()).first else { return }

        process(l: lEyeHitTest, r: rEyeHitTest)
    }

    private let buffer = CircularBuffer<SCNVector3>()

    fileprivate func process(l: SCNHitTestResult, r: SCNHitTestResult) {
        let middle = (l.worldCoordinates + r.worldCoordinates) / 2
        focusNode.position = middle

        buffer + view.projectPoint(middle)
```

```
        if buffer.isFull {
            onNextPoint?(buffer.average.cgPoint())
        }
    }

    private func sendFaceEvents(for anchor: ARFaceAnchor) {
        anchor.faceEvents.forEach { event in
            onFaceEvent?(event)
        }
    }
}

extension FaceTracker {
    typealias Gaze = (start: SCNVector3, end: SCNVector3)

    fileprivate func hitTest(node: SCNNode, gaze: Gaze) ->
[SCNHitTestResult] {
        let start = view.scene.rootNode.convertPosition(gaze.start, to:
node)
        let end = view.scene.rootNode.convertPosition(gaze.end, to: node)

        return node.hitTestWithSegment(from: start, to: end, options:
hitTestOptions())
    }

    fileprivate func hitTestOptions() -> [String: Any]? {
        return [SCNHitTestOption.backFaceCulling.rawValue: false,
                SCNHitTestOption.searchMode.rawValue:
SCNHitTestSearchMode.all.rawValue,
                SCNHitTestOption.ignoreChildNodes.rawValue: false,
                SCNHitTestOption.ignoreHiddenNodes.rawValue: false]
    }

    fileprivate func leftEyeGaze() -> Gaze {
        if let anchor = currentFaceAnchor, let node = currentFaceNode {
            let start = node.convertPosition(SCNVector3(m:
anchor.leftEyeTransform), to: nil)
            let end = node.convertPosition(SCNVector3(anchor.lookAtPoint),
to: nil)

            return (start, end)
        }

        let start = head.leftEye.worldPosition
        let end = head.leftEyeEnd.worldPosition

        return (start, end)
    }

    fileprivate func rightEyeGaze() -> Gaze {
        if let anchor = currentFaceAnchor, let node = currentFaceNode {
            let start = node.convertPosition(SCNVector3(m:
anchor.rightEyeTransform), to: nil)
            let end = node.convertPosition(SCNVector3(anchor.lookAtPoint),
to: nil)

            return (start, end)
        }
```

```
        let start = head.rightEye.worldPosition
        let end = head.rightEyeEnd.worldPosition

        return (start, end)
    }
}

extension CGPoint {
    func adjustedToScreenBounds() -> CGPoint {
        let bounds = UIScreen.main.bounds
        let newX = max(min(x, bounds.width), 0)
        let newY = max(min(y, bounds.height), 0)

        return CGPoint(x: newX, y: newY)
    }
}

private extension ARFaceAnchor {
    var faceEvents: Set<FaceExpressionEvent> {
        var events: Set<FaceExpressionEvent> = []

        if let value = blendShapes[.mouthSmileLeft]?.floatValue, value >
0.5 {
            events.insert(.smile)
        }

        if let value = blendShapes[.mouthSmileRight]?.floatValue, value >
0.5 {
            events.insert(.smile)
        }

        if let value = blendShapes[.eyeBlinkLeft]?.floatValue, value > 0.5
{
            events.insert(.blinkLeft)
        }

        if let value = blendShapes[.eyeBlinkRight]?.floatValue, value > 0.5
{
            events.insert(.blinkRight)
        }

        return events
    }
}
```

ДОДАТОК Б

Слайди презентації

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

АТЕСТАЦІЙНА РОБОТА МАГІСТРА

Дослідження методів управління складними об'єктами шляхом реалізації контролю за поглядом людини

Виконав:
ст. гр. ІПЗм-18-4
Радченко Г.С

Науковий керівник:
доц. Лановий О.Ф.

Мета роботи

- Дослідження методів управління складними об’єктами, основні методи, моделі, алгоритми контролю за поглядом людини та побудувати зручний механізм управління графічним інтерфейсом.
- Програмна реалізація мобільного додатку, що дозволить користувачу працювати із складними графічними елементами поглядом.

Аналіз предметної галузі

- Основною відмінністю складних об’єктів є наявність у них значної кількості властивостей на відміну від простих об’єктів.
- Основним засобом впливу на властивості об’єкта в комп’ютерних системах є курсор мишки та пункти меню, що дозволяють змінювати ці властивості.
- Існуючі програмні рішення є проприєтарними
- Відсутність підтримки існуючих продуктів
- Відсутність рішень на мобільних пристроях

3

Постановка задачі

- Дослідити методи машинного зору, доповненої реальності та машинного навчання
- Реалізувати програмну систему, що задовольняє наступним вимогам:
  - Розпізнавання обличчя то погляду користувача
  - Керування складними об’єктами шляхом аналізу погляду
  - Розпізнавання складних об’єктів методами машинного навчання
  - Керування властивостями складного об’єкта
- Винести моделі та операції для роботи з доповненою реальністю та нейромережою у окремі програмні модулі

4

Варіанти використання методу

- Керування інтерфейсом програми
- Полегшення використання існуючих систем для людей із обмеженними можливостями
- Революційний досвід у іграх
- Аналіз використання графічних елементів. Побудова теплових карт.
- Аналіз взаємодії користувача із рекламою

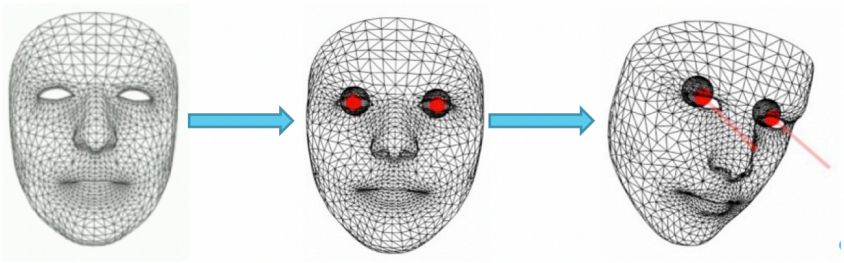
5

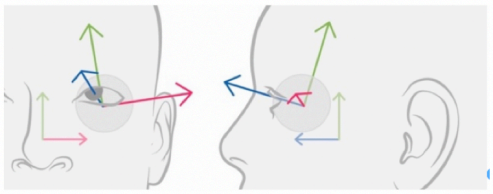
Огляд аналогів

- FaceDetector
  - Використовує веб камеру
  - Просте для використання
  - Немає аналізу даних
  - Працює тільки із браузером Google Chrome
- GazePointer
  - Просте для використання
  - Тільки для Windows
  - Немає підтримки
- PyGaze
  - Потужна система аналізу даних
  - Тільки для десктопів
- TurkerGaze
  - Основні функції аналізу даних
  - Тільки для Linux

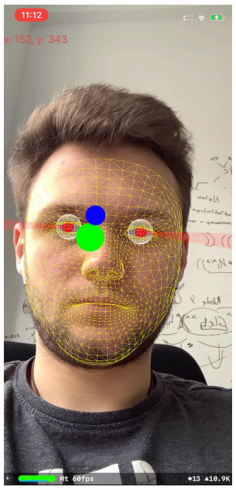
6

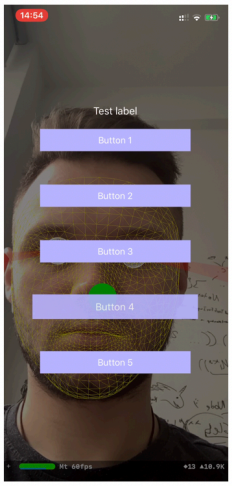
Сітка обличчя у доповненій реальності

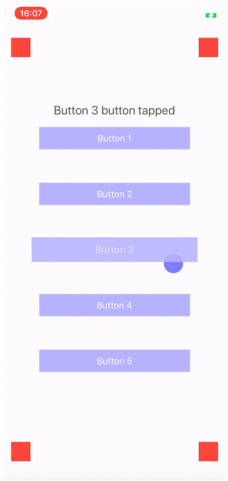




Інтерфейс продукту

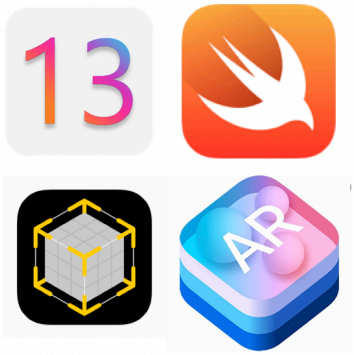




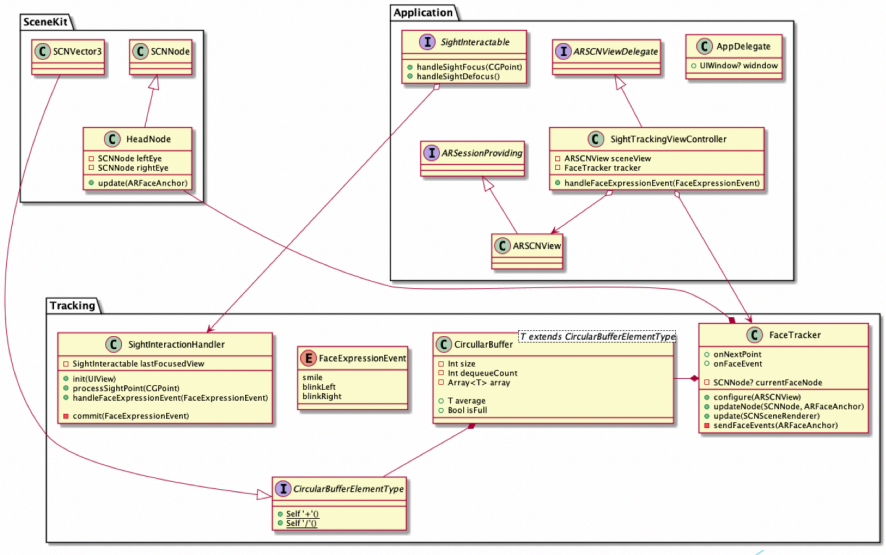


Стек технологій

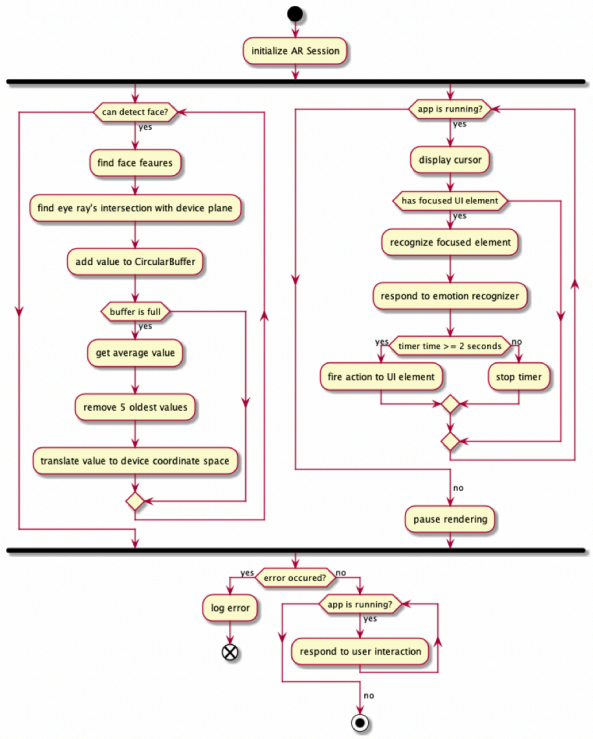
- iOS 13
- Swift
- ARKit 3.5
- Foundation, UIKit
- Swift PM



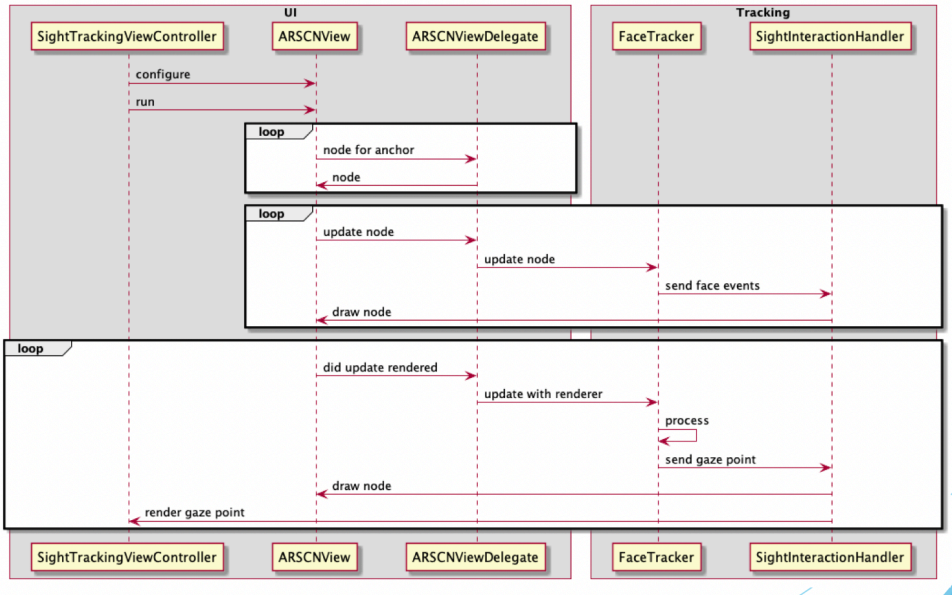
Діаграма класів



Діаграма діяльності системи



Діаграма послідовності системи



Висновки

- Проведено дослідження методів управління складними об’єктами шляхом реалізації контролю за поглядом людини
- Розроблений метод придатний для використання
- Вирішено завдання відстеження погляду користувача, а також емоцій, завдяки яким можна керувати складним графічним інтерфейсом
- Швидкий алгоритм аналізу погляду користувача, застосування нової технології доповненої реальності, машинного зору, синхронізація та калібрування обох технологій
- Розроблене програмне забезпечення задовольняє усім вимогам поставленої задачі

ДОДАТОК В

Апробація результатів роботи. Наукова стаття

User gaze tracking with Computer Vision and Linear Algebra

Nikita Fedosenko
Kharkiv National University of Radio Electronics
Kharkiv, Ukraine
nikita.fedosenko@nure.ua

Hlib Radchenko
Kharkiv National University of Radio Electronics
Kharkiv, Ukraine
hlib.radchenko@nure.ua

***Abstract*—Gaze tracking is an important and complicated problem in computer vision. As augmented and virtual reality system becoming more popular, there is a need in an accurate way of estimating gaze for providing more smooth interaction between human and computer systems. In this paper, we explore and implement one of the methods for gaze tracking, using the IOS platform.**

Keywords—Eye tracking, gaze tracking, gaze estimation

INTRODUCTION

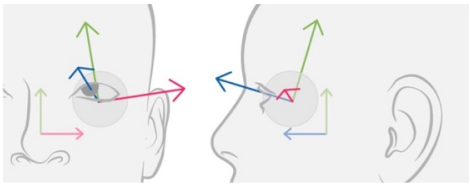
The main idea is to create such a technology/algorithm, which will allow to track user gaze and attention on a specific device based on iOS operating system. There are a lot of libraries that provides some required atomic functionality, but nowadays there is no any known algorithm, that will provide possibility to track and analyze user attention while they are using their devices.

To achieve the main goal, let's analyze existing technologies and frameworks to decide the way how to implement such idea.

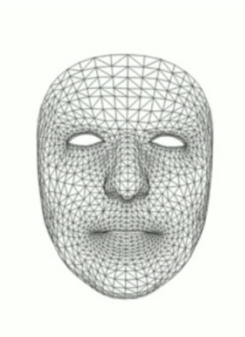
First of all, we must analyze and track user face using front camera. Then, we must detect user eyes to find his pupils. With existing information about user pupils, we must somehow calculate vectors of user's sight. After all, to achieve best accuracy, we must measure approximate distance from device to user head/eyes. Taking all above into consideration, we must analyze device position in real world and create device plane, create an equations with existing user-sight vectors to detect intersection points of user sight on a device plane. After all, we can convert calculated intersection points into device screen related points (device internal coordinate space). With information about user gaze, we can analyze information, make it smooth and accurate using existing math approaches.

IMPLEMENTATION

In 2016, the company called Apple, introduced a new revolutionary technology ARKit. This tool allows programmers to create augmented reality user experience using Apple devices. Later, Apple has introduced an extension to this technology to analyze user face and face feature points with modern True Depth cameras, installed on latest devices. The basic idea of this technology is to analyze real environment, tracking movement of device, camera and real objects. With modern cameras, devices can measure the distance to real objects very accurate. We can use face-tracking feature of this frameworks to get information about user's head location in device augmented reality coordinate space.



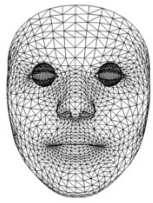
The information about face location is provided as transformation matrix with four rows and four columns. This matrix determines the distance (translation), the rotation an the scale of user face in real world according to local coordinate space. The information about face features (mouth, nose, eyebrows etc) is provided as a dictionary. Each key is a key for a specific face feature. Each value of this dictionary is a value between 0 and 1 determines, the accuracy of tracking single feature. All this information combined, presented as a 3D face mesh.



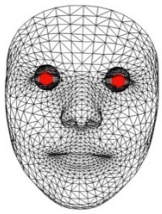
With existing information about user face location and transformation, we can calculate the approximate

location of eyes. The next step is to find pupils. This problem can be solved with Computer Vision tools. Apple provides Vision frameworks for such tasks. It applies computer vision algorithms to perform a variety of tasks on input images and videos. To achieve the goal of finding eye's pupils, there are a lot of methods presented. The most suitable algorithm for our purpose is CHT (Circle Hough Transform). This algorithm provides the possibility to detect circular objects in a digital image. The circle Hough Transform is a feature extraction technique for detecting circles. The purpose of the technique is to find circles in imperfect image inputs.

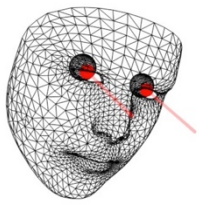
Human eyes has the radius of 12 mm, so our algorithm can easily predict eyes location having face mash and face transformation. We can project eye's spheres to augmented reality scene.



After finding pupils, we can also add them to augmented reality scene. By the way, human eye iris has the radius of 6 mm, so we can project it very accurate.



Having all required initial information, we can calculate gaze vectors, starting from the center of an eye sphere, crossing the center of a pupil.



The next step is to calculate distance between user face and real device(smartphone, tablet or other). Visual Odometry and True Depth camera embedded to Apple ARKit framework can provide the information about this distance. To create a correspondence between real and virtual spaces, ARKit uses a technique called visual-inertial odometry. This process combines information from the iOS device's motion sensing hardware with computer vision analysis of the scene visible to the device's camera. ARKit recognizes notable features in the scene image, tracks differences in the positions of those features across video frames, and compares that information with motion sensing data. The result is a high-precision model of the device's position and motion.

Apple cameras are based on SLAM(Simultaneous Localization and Mapping) technology. First, let's start with a basic explanation of how computer vision is used here. The iPhone's camera takes a picture which produces an image, and ARKit identifies features within that image. It then takes another picture, and again identifies the features. It then compares the first image to the second, and does a number of calculations to determine the change in the relationships between features. Doing this over and over as the camera continues to produce images - the iPhone can "track" the known features. We'll come back to this.

When you see things with your eyes, your brain does some instant calculation on the difference in the view from your right eye to your left eye and determines depth - the distance to different objects in your field of view. Now, there are objects that you know the approximate size of, and that information is used as well to judge the approximate scale of things around you - these are the "known features" being tracked. The important thing is your brain doing that calculation on the difference in the two images. Essentially this is triangulation - your right eye sees something in one way, your left eye sees it in a slightly different way, your brain knows the distance between your eyes, the approximate angle of focus, and the difference in the images, tracks some known features, and can therefore calculate approximate distances and understand the 3D space.

When an iPhone camera is turned on, it doesn't have two different images with which to calculate distances. However, a moment after the first image is taken it does have a second image. Thanks to data from the iPhone accelerometer sensors, it can also estimate the difference - from the first image to the second - of the iPhone camera's 3D position and aim. Now we go back to those "known features" being tracked. For each image the iPhone doesn't just do this for a single feature, it maps as many features as it can. Aside from doing the triangulation on each of the features in the images, it also does a comparison between the differences in each feature's relationship to other features in the image. So now, like your brain, the iPhone has two different views of something, knows the approximate angles of focus, knows the distance between the lens position, is tracking known features and their relationship to each other. From this, the iPhone can get a very good approximation of how each feature is positioned in space with relation to the other features, essentially producing a 3D mapping of the space.

So we can rely on information provided by ARKit to perform our basic calculations based on augmented reality scene.

To detect the position of user's gaze, we must create a virtual plane, according to the device position in real world. It's already provided in ARKit, using CoreMotion framework under the hood. The position of a device is presented by mentioned above translation matrix with information about rotation and translation of a device.

Having this plane, two vectors of user eyes, we can calculate the intersection of this vectors with device plane and approximate them.

First of all, let's create the equation of a line by two points in space. We have to points $A(x_1,y_1,z_1)$ and $B(x_2,y_2,z_2)$. The equation is:

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1} = \frac{z - z_1}{z_2 - z_1}$$

Then, let's create the equation of a plane. Virtual device plane has it's transformation matrix with the information of an angle, so we can create a normal vector, by rotating basic vector $\underline{n}(a, b, c)$ = (1, 1, 1) with device plane angle transformation. To create an equation of a plane, we need also a point, associated with this plane. We can take point P(0, 0, 0) as a plane point, because our coordinate space is associated with device camera, using camera position as a zero point. The equation of a plane is:

$$a(x - x_1) + b(y - y_1) + c(z - z_1) = 0$$

Our initial point has zero values, so we can simplify our equation:

$$ax + by + cz = 0$$

To find the intersection point, let's calculate an equation system. This point must satisfy the equation system.

$$\begin{cases} \dfrac{x - x_1}{x_2 - x_1} = \dfrac{y - y_1}{y_2 - y_1} = \dfrac{z - z_1}{z_2 - z_1} \\ ax + by + cz = 0 \end{cases}$$

This equation is a system of 3 equations with three unknowns, so it can be easily solved using different approaches (such as Gauss or Cramer methods). The solution of this system (if exist) is a point $M(x_3, y_3, z_3)$, which represents the intersection point between device plane and line.

Having this solution, we can find two points, which represent intersections between device plane and gaze vectors of left and right eye. Then we can find the middle point, which is the final user gaze point projected to a device plane.

The final step is to convert global mid-intersection point presented in augmented reality scene to a local coordinates of a device screen. To achieve this, native approach can be applied, using hit testing of a view.

CONCLUSIONS

After all calculations, converted points are calculated and processed to render user gaze marker to a device view.

Taking all above mentioned into consideration, the lightweight and accurate approach of tracking user gaze is described. This method can provide a variety of different possibilities to developers and users. One example is to create such a frameworks, which will allow to control application UI using only user gaze. This approach will allow disabled people to use their smartphones without having any problems. The next possibility is to build a revolutionary new user experience in games. For example, people can play local or multiplayer games using only their eyes. And the final technique is tracking user gaze to collect and analyze data for creating heat maps of applications, or for improving UI and user experience. This feature is an interesting way how to provide an absolutely new way of controlling devices for many people.

REFERENCES

G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*

J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

K. Elissa, "Title of paper if known," unpublished.

R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

ДОДАТОК Г

Акт прийому наукової статті

Submission Summary

Conference Name
2020 IEEE 15th International Conference on Computer Sciences and Information Technologies (CSIT)

Track Name
Software engineering

Paper ID
52

Paper Title
User gaze tracking with Computer Vision and Linear Algebra

Abstract
Gaze tracking is an important and complicated problem in computer vision. As augmented and virtual reality system becoming more popular, there is a need in an accurate way of estimating gaze for providing more smooth interaction between human and computer systems. In this paper, we explore and implement one of the methods for gaze tracking, using the IOS platform.

Created on
24.04.2020, 15:44:24

Last Modified
24.04.2020, 15:44:24

Authors
Nikita Fedosenko (Kharkiv National University of Radio Electronics) <nikita.fedosenko@nure.ua>
Hlib Radchenko (Kharkiv National University of Radio Electronics) <hlib.radchenko@nure.ua>

Submission Files
statya.docx (226.5 Kb, 24.04.2020, 15:44:15)