iOS Image Overlay Swift

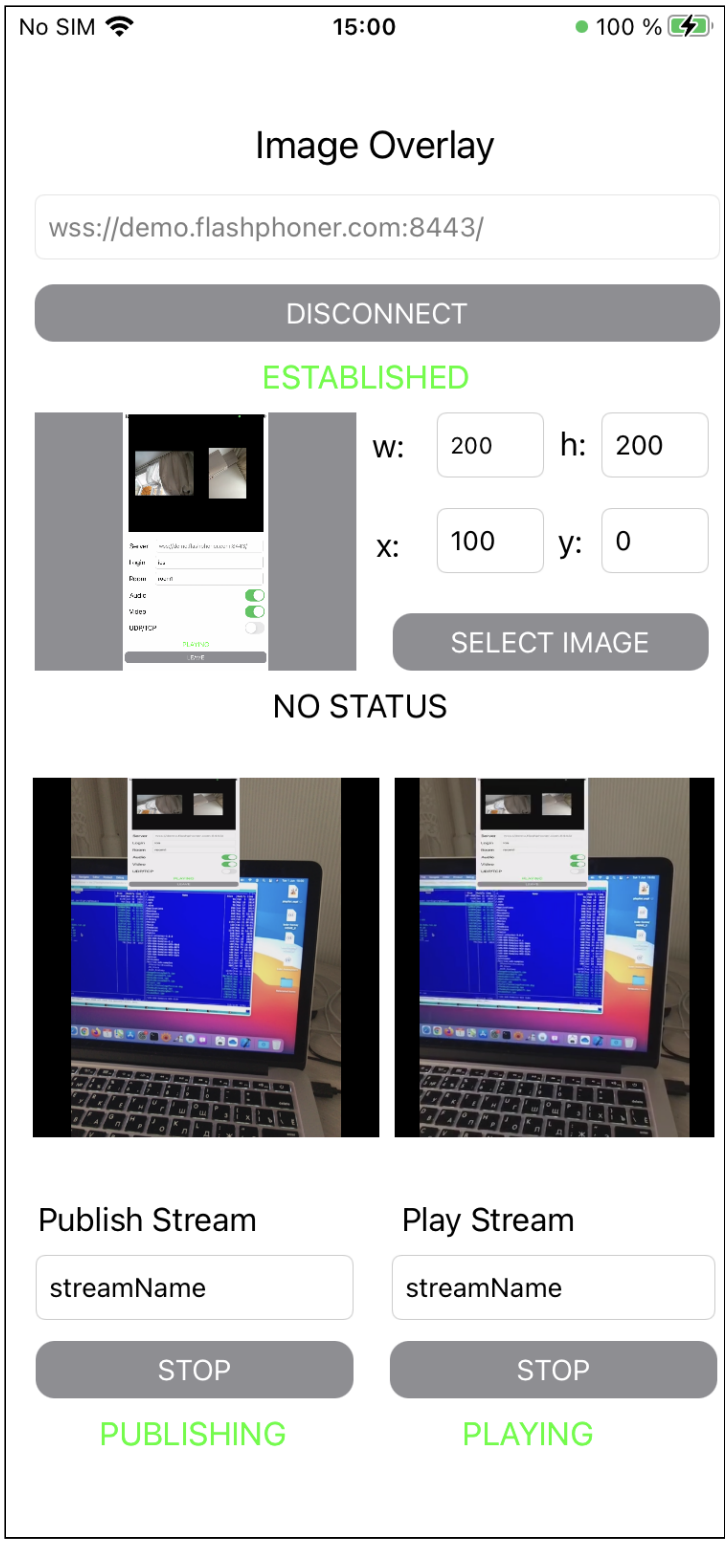
Пример масштабирования публикуемого изображения и добавления PNG картинки

Данный пример демонстрирует масштабирование публикуемого изображения (увеличение и уменьшение щипком двумя пальцами), а также наложение картинки из галереи устройства.

На скриншоте ниже в поток добавлена картинка с указанием ее размеров и положения в кадре.

Поля ввода:

- `WCS URL` - адрес WCS сервера
- `w` - ширина накладываемой картинки
- `h` - высота накладываемой картинки
- `x` - положение верхнего левого угла картинки в кадре по горизонтальной оси
- `y` - положение верхнего левого угла картинки в кадре по вертикальной оси
- `Select image` - кнопка выбора картинки из галереи



Картинка и ее расположение могут меняться на лету, во время публикации.

Работа с кодом примера

Для разбора кода возьмем версию примера ImageOverlaySwift, которая доступна для скачивания на GitHub:

- `ImageOverlayViewController` - класс основного вида приложения (файл имплементации ImageOverlayViewController.swift)
- `CameraVideoCapturer` - класс, реализующий захват и обработку видео (файл имплементации CameraVideoCapturer.swift)

1. Импорт API

code

```
import FPWCSApi2Swift
```

2. Инициализация класса для захвата и обработки видео

code

```
var capturer: CameraVideoCapturer = CameraVideoCapturer()
```

3. Создание сессии и подключение к серверу

`WCSSession`, `WCSSession.connect` code

В параметрах сессии указываются:

- URL WCS-сервера
- имя серверного REST hook приложения `defaultApp`

```
@IBAction func connectPressed(_ sender: Any) {
    changeViewState(connectButton, false)
    if (connectButton.title(for: .normal) == "CONNECT") {
        if (session == nil) {
            let options = FPWCSApi2SessionOptions()
            options.urlServer = urlField.text
            options.appKey = "defaultApp"
            do {
                try session = WCSSession(options)
            } catch {
                print(error)
            }
        }
        ...
        changeViewState(urlField, false)
        session?.connect()
    } else {
        session?.disconnect()
    }

}
```

4. Публикация видеопотока

`WCSSession.createStream`, `WCSStream.publish` code

Методу `createStream` передаются параметры:

- имя публикуемого потока
- вид для локального отображения
- объект для захвата видео

```
@IBAction func publishPressed(_ sender: Any) {
    changeViewState(publishButton,false)
    if (publishButton.title(for: .normal) == "PUBLISH") {
        let options = FPWCSApi2StreamOptions()
        options.name = publishName.text
        options.display = localDisplay.videoView
        options.constraints = FPWCSApi2MediaConstraints(audio: true,
videoCapturer: capturer);
        do {
        publishStream = try session!.createStream(options)
        } catch {
            print(error);
        }
        ...
        do {
            try publishStream?.publish()
            capturer.startCapture()
        } catch {
            print(error);
        }
    } else {
        do {
            try publishStream?.stop();
        } catch {
            print(error);
        }

    }

}
```

5. Воспроизведение видеопотока

`WCSSession.createStream`, `WCSStream.play` code

Методу `createStream` передаются параметры:

- имя воспроизводимого потока
- вид для отображения потока

```
@IBAction func playPressed(_ sender: Any) {
    changeViewState(playButton,false)
    if (playButton.title(for: .normal) == "PLAY") {
        let options = FPWCSApi2StreamOptions()
        options.name = playName.text;
        options.display = remoteDisplay.videoView;
        do {
        playStream = try session!.createStream(options)
        } catch {
            print(error)
        }
        ...
        do {
            try playStream?.play()
        } catch {
            print(error);
        }
    } else{
        do {
            try playStream?.stop();
        } catch {
            print(error);
        }
    }
}
```

6. Остановка воспроизведения видеопотока

`WCSStream.stop` code

```
@IBAction func playPressed(_ sender: Any) {
    changeViewState(playButton,false)
    if (playButton.title(for: .normal) == "PLAY") {
        ...
    } else{
        do {
            try playStream?.stop();
        } catch {
            print(error);
        }
    }
}
```

7. Остановка публикации видеопотока

`WCSStream.stop` code

```
@IBAction func publishPressed(_ sender: Any) {
    changeViewState(publishButton,false)
    if (publishButton.title(for: .normal) == "PUBLISH") {
        ...
    } else {
        do {
```

```
            try publishStream?.stop();
        } catch {
            print(error);
        }

    }

}
```

8. Вызов функции масштабирования видео щипком по виду для локального отображения потока

code

```
@IBAction func pinchOnLocalDisplay(_ sender: UIPinchGestureRecognizer) {
    if sender.state == .changed {
        self.capturer.scale(velocity: sender.velocity)
    }
}
```

9. Выбор картинки из галереи и вызов функции наложения картинки

code

```
@IBAction func selectImagePressed(_ sender: Any) {
    imagePicker.allowsEditing = false
    imagePicker.sourceType = .photoLibrary
    DispatchQueue.main.async {
        self.present(self.imagePicker, animated: true, completion: nil)
    }
}

func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any])
{
    guard let image = info[.originalImage] as? UIImage else {
        return;
    }

    selectedImage = image
    imageView.image = selectedImage
    updateOverlayImage()

    DispatchQueue.main.async {
        picker.dismiss(animated: true, completion: nil)
    }
}
```

10. Масштабирование выбранной картинки, задание ее координат и вызов функции наложения картинки

code

```
func updateOverlayImage() {
    if let selectedImage = selectedImage {
        let resizeImage = resize((selectedImage.cgImage)!,
selectedImage.imageOrientation)

        let overlayImage = CIImage.init(cgImage: (resizeImage)!)
        let overX = CGFloat(Int(overlayX.text ?? "0") ?? 0)
        let overY = CGFloat(Int(overlayY.text ?? "0") ?? 0)
        let movedImage = overlayImage.oriented(.left).transformed(by:
CGAffineTransform(translationX: overY, y: overX))
        capturer.updateOverlayImage(movedImage)
    } else {
        capturer.overlayImage = nil
        return
    }
}
```

11. Реализация масштабирования видео

code

```
func scale(velocity: CGFloat) {
    guard let device = self.device else { return }

    let maxZoomFactor = device.activeFormat.videoMaxZoomFactor
    let pinchVelocityDividerFactor: CGFloat = 15


    do {
        try device.lockForConfiguration()
        defer { device.unlockForConfiguration() }

        let desiredZoomFactor = device.videoZoomFactor + atan2(velocity,
pinchVelocityDividerFactor)
        device.videoZoomFactor = max(1.0, min(desiredZoomFactor,
maxZoomFactor))
    } catch {
        print(error)
    }
}
```

12. Реализация наложения картинки

code

```
let pixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer)
...
if (overlayImage != nil) {
    let inputImage = CIImage.init(cvImageBuffer: pixelBuffer!);
    let combinedFilter = CIFilter(name: "CISourceOverCompositing")!
```

```
    combinedFilter.setValue(inputImage, forKey: "inputBackgroundImage")
    combinedFilter.setValue(overlayImage, forKey: "inputImage")

    let outputImage = combinedFilter.outputImage!
    let tmpcontext = CIContext(options: nil)
    tmpcontext.render(outputImage, to: pixelBuffer!, bounds:
outputImage.extent, colorSpace: CGColorSpaceCreateDeviceRGB())

}
```