Journal of Korea Game Society 2020 Aug; 20(4): 11-20
http://dx.doi.org/10.7583/JKGS.2020.20.4.11

게임 프로그래밍

React Native와 Unity3D를 활용한 크로마키 기반 이미지 합성 모바일 앱 개발

김승준, 서범주, 조성현
홍익대학교 일반대학원 게임학과
kijlk@naver.com, {bseo, scho}@hongik.ac.kr

Development of a Mobile App Combining React Native and Unity3D for Chromakey-based Image Composition

Seung-Jun Kim, Beom-Joo Seo, Sung-Hyun Cho
Dept. of Games, General Graduate School, Hongik University

요 약

모바일 시장에서는 앱을 빠르게 개발하여 시장의 평가를 받는 것이 중요하다. 하지만 소규모 회사에서 단편화된 모바일 환경에 대응하여 제품을 신속하게 개발하고 배포하는 것은 어려운 일이다. 본 연구에서는 높은 수준의 기능과 성능을 요구하는 모바일 앱 개발 시에 크로스 플랫폼 솔루션인 React Native와 Unity3D를 동시에 활용하여 제작한 통합 개발 결과물이 요구 조건을 충족할 수 있음을 보인다. 또한 크로스 플랫폼을 통합한 개발 방식이 모바일 앱의 개발 기간을 단축하고 개발 비용을 절감할 수 있음도 보여준다.

ABSTRACT

In the rapidly changing mobile app market, it is crucial to develop a good idea quickly and receive its market evaluation. For a small-sized company, however, it is very challenging to rapidly develop and deploy their products in response to highly fragmented mobile environments. This article demonstrates that our integrated development environment using both React Native and Unity3D when developing a mobile app achieves a high level of functionality and performance requirements successfully. Moreover, this integrated environment helps reduce development costs and shorten development time.

Keywords : Cross Platform(크로스 플랫폼), Mobile App(모바일 앱), React Native, Unity3D, Chromakey(크로마키), Image Processing(이미지 처리)

Received: Jun. 19. 2020 Revised: Aug. 07. 2020
Accepted: Aug. 09. 2020
Corresponding Author: Beom-Joo Seo(Hongik University)
E-mail: bseo@hongik.ac.kr

ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

모바일 게임 개발자들은 빠르게 변화하는 모바일 시장에 대응하여 게임을 빠르게 구현하고 출시하는 것을 목표로 하고 있으나, 이를 실제 개발 환경에 적용할 때 산적해 있는 다양한 현실적인 문제들을 극복해야 한다. 최근 One Source Multi Use(OSMU)라는 개념으로 다양한 플랫폼에 대응하여 수많은 모바일 기기 종류에 적용 가능한 크로스 플랫폼 개발 기술들이 도입되어 개발자들의 부담을 경감시키고 있다[1]. 또한 개발자가 접근하기 쉬운 성숙한 웹기반 기술을 적극적으로 활용한 Progressive Web Apps (PWA) 기술은 기존 네이티브 솔루션, 크로스 플랫폼 솔루션과 경쟁하고 있으나 성능이 기대에 미치지 못해 활용범위가 정체되어 있다[2].

본 연구에서는 이러한 크로스 플랫폼 기술 중에서 최근 각광받고 있는 React Native와 Unity3D를 적용하여 모바일 앱을 제작할 때, 기능과 성능에서 발생가능한 문제점들이 무엇이며 이를 해결하기 위한 어떤 방안들이 있는지 실제 개발 사례를 통해서 알아본다.

이를 위해 기능 및 성능 측면에서 높은 수준의 요구사항을 갖는 모바일 앱을 개발하였다. 이 앱을 안드로이드에 적용하여 해당 기기들에서 기능성, 안정성, 네트워크 성능을 측정하여 다양한 안드로이드 기기에서 실제 운용가능성 여부를 알아본다. 본 연구에서는 동화책 편집을 위한 모바일 앱을 대상으로 크로마키 처리를 통한 실시간 이미지 합성 기능을 구현하고자 한다.

논문의 구성은 다음과 같다. 2장에서는 크로스 플랫폼 기술들에 대하여 소개하고, 기술 개발 대상인 크로마키 처리 기법에 대해 알아본다. 3장에서는 본 연구에서 개발하는 서비스 플랫폼을 소개하고, 본 연구의 접근 방법 및 요구사항 분석을 기술한다. 4장에서는 Unity3D로 구현된 크로마키 처리 기법 및 해당 기법을 React Native와 연동하는 모델, 전용 이미지 캐싱 방식 등을 기술한다. 5장에서는 개발된 모바일 앱의 기능성 테스트 및 크로마키 처리 기반 이미지 데이터의 업로드 응답시간 측정을 통하여 실제 애플리케이션에 활용 가능성을 타진한다. 마지막으로 6장에서는 연구 결과를 요약한다.

2. 관련 연구

2.1 크로스 플랫폼 기술

크로스 플랫폼 개발을 지원하는 다양한 개발 환경은 오픈소스 형태로 혹은 판매용 제품 형태로 나타나고 있다. 널리 사용되는 제품들로는 Ionic, PhoneGap, React Native, Xamarin 등이 있다[3]. Ionic이나 PhoneGap은 웹과 전용 기술을 혼합한 형태로 제공하고 있으며, React Native는 Javascript를 활용하여 플랫폼별로 제공되는 네이티브 인터페이스 컴포넌트를 브리지 형태로 연결하여 사용한다. 반면 Xamarin은 C#으로 개발된 코드를 크로스 컴파일 과정을 거쳐 네이티브 바이너리 코드로 만든다.

크로스 플랫폼을 제공하는 다른 접근 방법으로 게임 엔진 플랫폼(Unity3D[4], Unreal Engine[5])을 활용하는 개발 환경이 있다. 이들 게임 엔진은 게임 제작에 특화된 솔루션을 제공하고 있으며, 모바일 게임 시장이 활성화되면서 모바일 기기를 위해 개발한 UI/UX 컴포넌트들이 일반 모바일 앱 제작에 활용되기 시작하면서 각광받고 있다. 그러나 게임로직 개발에 특화된 솔루션이기 때문에 성능 면에서 개선될 여지가 많다.

웹 기술을 모바일 앱 환경에 그대로 적용할 수 있는 WebView[6] 기술은 웹에서 구현한 UI/UX 컴포넌트를 추가 개발 없이 바로 적용할 수 있는 방법으로 많은 기대를 받았다. 그러나 성능이 만족스럽지 않아 제한된 환경에서만 활용되고 있다.

본 연구에서는 웹 기술을 기반으로 함으로써 새로운 기술 도입의 장벽을 줄이고 동시에 성능 저하 요인을 줄이는 방안으로 Facebook에서 개발한 React Native 프레임워크를 기반으로 한 모바일

앱 개발 방법론의 실제 활용 가능성을 알아보고자 한다. React Native는 웹 기술을 확장하였기 때문에 기존 웹기반 프로그래밍에 익숙한 개발자를 수용할 수 있으며, Virtual DOM과 같이 렌더링 성능 개선 요소를 도입하여 모바일 환경에서 네이티브 애플리케이션 수준의 성능을 만족할 수 있는 것으로 기대되고 있다[7]. 이 프레임워크는 2015년 Facebook에서 오픈소스로 배포하였으며, 안드로이드, iOS, UWP 개발환경을 지원한다. 언리얼 엔진과 동일하게 최근 게임엔진에서 적용하고 있는 컴포넌트 기반 설계 기법을 도입하고 있으며, 이미 존재하는 웹 기술 기반으로 Javascript, JSX 등을 사용하여 빠른 UI/UX 개발이 가능하다.

본 연구에서는 UI/UX 컴포넌트 개발에 유리한 웹 기술은 React Native 플랫폼을 활용하여 개발하고, React Native에서 매우 제한적으로 제공하는 실시간 이미지 처리와 같은 특화 기능들은 Unity3D를 활용하여 개발하고자 한다[8].

2.2 매팅 기술

매팅(Matting)은 사각형의 배경 이미지로부터 비정형 전경 이미지를 분리한 후 전경 이미지를 다른 배경과 합성하는 방법이다.

일반적으로 전경과 배경을 분리하는 문제는 매우 어려운 문제이다. 이를 손쉽게 해결하는 방법으로 고정된 색상(일반적으로 파란색)을 배경으로 설정하여 전경을 분리한 후, 새로운 배경 이미지와 합성하는 방법으로 블루스크린매팅 혹은 크로마키 처리 방법이 널리 사용된다[9]. 이 방법은 이미지에서 배경으로 설정된 색이 아닌 색들을 출력하는 간단한 기법을 사용하여 원하는 객체를 손쉽게 추출한다. 크로마키 기법은 조명에 매우 민감하기 때문에 그림자가 생기는 경우 오류가 발생하기 때문에 적절한 조명 시설과 단일 배경이 갖춰진 환경에서 작업해야 한다.

최근에는 이미지 분리 기법이 진화하여 딥러닝을 기반으로 영상에서 물체의 배경을 자동으로 분리하는 기술이 도입되고 있다[10].

본 연구에서는 모바일 앱 개발에 상대적으로 용이한 크로마키를 기반으로 분리한 전경 객체를 동화의 배경에 합성하는 방법을 사용한다.

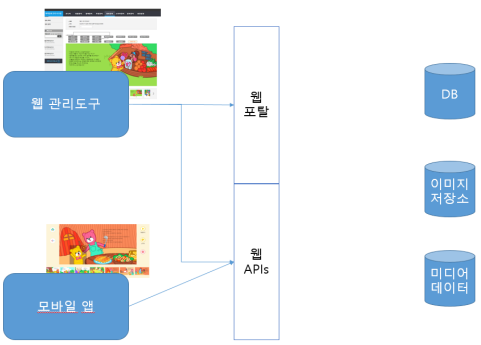
3. 목표 시스템

3.1 시스템 개요

본 연구에서 개발하는 모바일 앱은 동화 편집/재생 서비스의 일부로서 전체 시스템은 [Fig.1]에서 제시하고 있다. 시스템은 동화 서비스를 전체 관리하는 웹 서비스 시스템과 Representational State Transfer (REST)[11] API를 통해 웹 서버와 통신하는 모바일 앱으로 구성된다. 웹 서비스에는 여러 개의 웹 기반 관리도구로 구성되어 있다. 동화 서비스를 제공하기 위해 동화를 온라인상에서 제작할 수 있는 동화 제작 시스템, 제작된 동화를 수정, 발간할 수 있는 동화 관리 시스템, 스티커 부착 등과 같은 부가 서비스 제공 시스템, 사용자 관리 시스템 등으로 구성된다.

동화 제작 시스템을 통해 제작된 동화에는 페이지를 구성하는 미디어 콘텐츠 정보들(이미지, 배경음악, 내레이션 음악 등)과 이들 콘텐츠를 활용하여 페이지에 색인화한 메타정보들로 구성되어 있다. 콘텐츠 정보들은 별도의 미디어 저장소에, 메타정보들은 미디어 저장소 내 해당 미디어를 접근할 수 있는 URI 형태의 식별자와 텍스트 정보들, 애니메이션 관련 정보 등 색인화 정보들을 JSON 형식으로 표현된다. JSON 형식의 메타정보는 자유롭게 동화책 정보를 편집할 수 있도록 DBMS 시스템에 색인화 되어 있다.

외부 프로그램이 웹 서버와 연동할 수 있도록 설계된 웹 API에는 미디어 데이터(이미지 포함)의 업로드와 다운로드 기능을 제공하는 미디어 API, 접속 관련 API, 동화 목록 검색 API, 동화 내 페이지별 메타정보를 획득하거나 갱신하는 기능을 제공하는 동화 페이지 API, 구매 관련 API 등 기본 기능들을 제공한다. 응답 결과는 JSON 혹은 미디어 형식으로 반환된다.



[Fig. 1] Overview of Our Fairy Tale Service

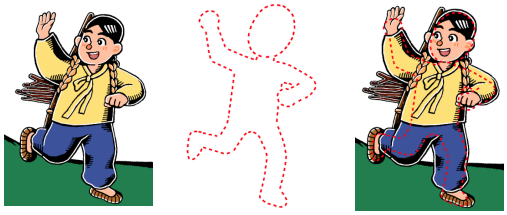
모바일 앱은 위에서 언급한 웹 API를 활용하여 동화 편집/재생 서비스를 제공한다. 로그인한 사용자는 구매한 동화 목록을 검색할 수 있으며, 구매한 동화를 자신이 원하는 형식으로 편집할 수 있고, 편집된 동화를 재생할 수 있다. 이때 편집 가능한 동화 페이지 내 요소들은 모바일 앱 사용자와 상호작용이 가능하다.

본 연구에서는 모바일 앱에서 크로마키 처리 기법을 활용하여 사용자가 자신이 원하는 이미지 객체를 기존 동화 페이지에 합성하고 동화를 재생하는 방법을 사용한다.

3.2 교체를 통한 이미지 합성

동화 페이지에서 교체할 이미지를 페이지에 등록하는 과정, 교체 작업을 위한 상호작용 모델, 크로마키 처리를 통해 교체 이미지를 추출하고 동화 페이지에서 이미지를 합성하는 과정을 기술한다.

동화 페이지에서 교체할 이미지는 모바일앱 사용자와 상호작용을 통해 원하는 이미지로 교체될 수 있다. 상호작용이 가능한 이미지는 동화 페이지 제작 시에 동화에 포함될 일반 이미지를 등록하는 과정과 동일하다. 이미지 등록 후 '점선가이드 이미지 등록' 버튼을 눌러 원본 이미지에 연동될 점선가이드 이미지를 지정하면 원본 이미지는 상호작용 이미지 객체로 자동 등록된다. 연동되는 점선가이드 이미지는 원본 이미지와 동일한 해상도를 가지며 동화 제작 시에만 보인다([Fig. 2] 참조).



(a) Original Image (b) Dotted Guide Image (c) Composed Interactable Image Object

[Fig. 2] Constructing Interactable Image Combining Original with Guide Images

모바일앱 사용자는 원본 이미지 교체를 위해 원본이미지를 선택하여 촬영 모드로 진입할 때 점선가이드 이미지를 볼 수 있다. 사용자는 삽입하고자 하는 객체를 점선가이드 그림에 맞추어 자세를 취하고 크로마키 처리를 통해 추출한다. 추출된 객체 이미지는 동화페이지의 원본이미지 위치에 교체되어 렌더링 된다.

본 연구에서 의미하는 이미지 합성은 실제 여러 개의 이미지를 합성하여 하나의 이미지로 합성하는 것이 아니라, 각기 다른 레이어에 존재하는 이미지 객체들을 층층이 쌓아 올려 하나의 합성된 이미지처럼 보이게 하는 효과를 지칭한다. 이미지 합성 과정에 대한 상세 설명은 4.1에서 기술한다.

3.3 모바일 앱 요구사항 분석

앞에서 기술한 바와 같이 본 논문에서 구현하고자 하는 모바일 앱은 서버와의 통신을 통해 사용자가 동화 내에서 상호작용 가능한 이미지들을 편집하여 자신만의 동화책을 만들 수 있는 기능을 제공하는 것을 목표로 한다. 이를 위해 모바일 앱에 필요한 기능과 성능의 요구사항은 다음과 같다.

모바일 앱 사용자는 기존 동화 페이지에서 상호작용이 가능한 객체 이미지를 교체할 수 있다. 이

때 교체되는 이미지 객체를 생성하기 위해 카메라에서 촬영한 이미지에서 후경 이미지를 제거하는 매팅 기법을 사용한다. 매팅 기법을 적용할 때 전경 이미지객체가 원하는 대로 분리되는지 실시간적으로 확인가능 해야 한다(**실시간성**). 원하는 전경 이미지 객체를 획득하면, 크로마키 처리된 이미지를 서버에 적정 시간 이내에 업로딩하고 그 이미지를 모바일 앱 환경에서 사용할 수 있도록 변환할 수 있어야 한다(**적정 업로드 응답시간**). 본 연구에서는 적정 응답시간은 3초로 설정하였다. 마지막으로 사용자 관리, 동화 보기 및 재생, 동화 페이지 편집 등의 동화 제작 앱으로서 보편적인 기능들이 모바일 기기에서 안정적으로 동작해야 한다(**모바일앱 기능성**).

3.4 개발 환경

일반적으로 모바일 앱 개발 시 모바일 운영체제에서 제공하는 개발 생태계를 사용한다. 하지만 OSMU 솔루션은 플랫폼별 기술 개발을 단일화하고, 제한된 예산과 짧은 개발 기간, 향후 운영체제별 편리한 유지보수 등을 가능하게 한다. Web 표준 기술로 널리 쓰이는 HTML5와 CSS3, Javascript를 기반으로 앱에서 요구하는 성능을 만족시키는 기술로 PWA 기술이 발전해오고 있으나, 아직 만족스러운 성능을 제공하지 못하기 때문에 모바일 앱 개발에 무리가 있다. 그래서 본 연구에서는 유연한 이식성과 성능을 제공할 수 있는 React Native와 Unity3D를 기본 개발 환경으로 선정하였다.

일반 모바일 앱 개발은 웹 기술을 기반으로 구축된 React Native 기술을 활용하였으며, 실시간 크로마키 처리 기술은 Unity3D를 기반으로 제작되었다. 모듈을 두 개의 플랫폼으로 나누어 제작하였기 때문에 이들 모듈을 통합 운영하는 기술 확보가 관건이다. 상세한 연동 방법은 4.2에서 기술한다.

3.5 동화책 제작

동화책 앱에서 사용되는 동화 제작 기술 및 그 저장 형식을 설명한다. 동화책을 꾸미는 기반 도구로 오픈 소스인 fabric.js[12]를 활용한다. fabric.js는 HTML5 Canvas를 활용한 이미지 조작 라이브러리로서 벡터 그래픽을 그리거나 이미지 기반 객체의 제어점을 사용하여 회전, 확대, 축소, 이동한 후에 캔버스 내에 자유롭게 배치한다. CSS를 이용한 애니메이션을 제공하며, 복잡한 수준의 애니메이션은 스크립트 프로그래밍을 통해 지원하고 있다. 이와 유사한 path.js, raphael.js 등이 있으나 본 연구에서는 다양한 객체들을 캔버스에 편리하게 배치할 수 있도록 구현된 fabric.js를 선정하였다.

이 라이브러리는 캔버스 (역)직렬화 기능을 제공하여 캔버스에 올린 객체 정보를 JSON 형태로 변환할 수 있다. 따라서 동화책 제작 완료 후 제작 내용을 JSON 포맷으로 변환한 후에 DBMS에 저장한다.

4. 모바일 앱 구현

본 장에서는 모바일 앱 구현에 사용된 기술들을 설명한다. Unity3D로 개발된 크로마키 처리 로직은 4.1에서, Unity3D로 개발한 컴포넌트와 React Native 모듈 연동 방식은 4.2에서, React Native에서 개발한 이미지 캐싱은 4.3에서 기술한다.

4.1 크로마키 처리

본 연구에서는 모바일 기기에 부착된 카메라를 통해 인식된 비디오 프레임으로부터 원하는 전경 객체를 분리하기 위해 크로마키 필터링 기법을 사용한다. 이 기법으로 원하는 배경 색상 및 명도, 채도를 실시간으로 설정하여 원하는 전경 객체가 시각적으로 추출되는지 확인하는 작업이 필요하다. 이러한 실시간성은 React Native에서 직접 제공하지 않기 때문에 Unity3D를 이용하여 구현하였다.

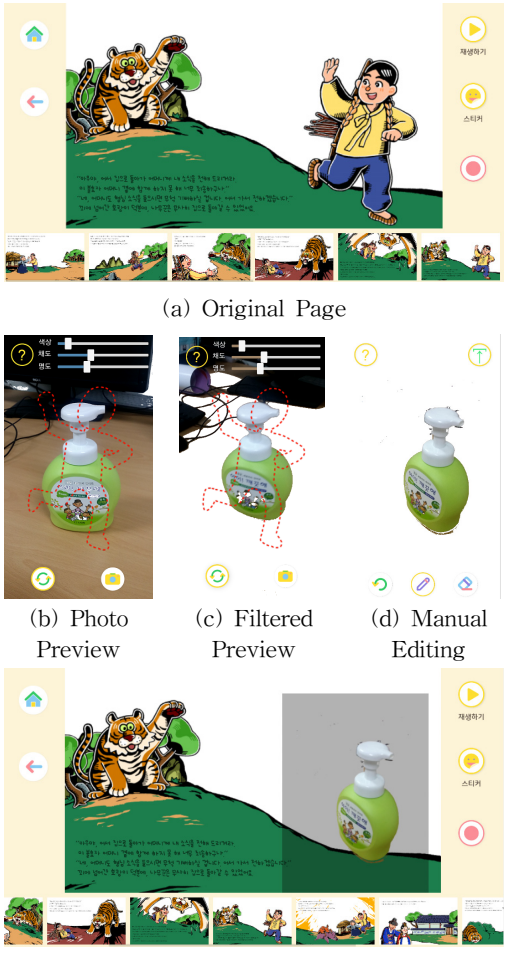
[Fig.3]은 이 절차를 간략하게 표현하고 있다. 먼저 [Fig.3]의 (a), (e)는 각각 동화 페이지에서 상호작용이 가능한 이미지에 분리된 전경 객체를

합성하기 전과 후에 찍은 스크린샷으로 React Native에서 개발되었다. (a)에서 상호작용 가능한 이미지를 선택하면 Unity3D로 개발된 (b), (c), (d) 화면 순으로 진입한다. 이때 상호작용 가능한 이미지에 연계된 점선가이드 이미지가 사진 촬영 화면에 나타나 사진 촬영에 도움을 준다([Fig.3](b) 참조). [Fig.3](c)에서는 크로마키 처리하고자 하는 배경색을 조정하여 원하는 전경 객체 추출여부를 실시간으로 보여주고 있다. 그림에서 하얀색은 크로마키 처리로 삭제된 배경이다. [Fig.3](d)은 삭제가 덜된 배경을 수작업을 통해 편집하는 장면을 보여준다. 최종 편집 완료 후 [Fig. 3](e)에서 보여주는 동화 페이지로 돌아올 때 크로마키 처리된 이미지는 서버에 업로드 된다. 업로드 된 이미지는 모바일 환경에 맞게 이미지 해상도가 변환되어 저장된다. 모바일 앱은 변환 결과물을 다운로드 받아 원본 이미지를 교체함으로써 사용자에게 합성된 이미지를 보여준다.

크로마키 처리시 실시간 크로마키 처리를 위해 모바일 앱에서 제공하는 GPU를 활용한 크로마키 쉐이더를 사용하여 구현하였다. 실시간 크로마키 처리는 촬영 중 원본 이미지 픽셀에 대한 원시 데이터 접근을 필요로 하는데, 이는 성능 저하의 요인이 된다. 따라서 전용 필터링 로직을 구현하는 대신 성능을 극대화하기 위해 WebGL에서 제공하는 쉐이더 기능을 활용하였다. 크로마키의 쉐이더 처리는 내장된 그래픽 카드의 기능을 활용함으로써 소프트웨어 기반 크로마키 필터링 처리보다 성능을 향상시킬 수 있다.

상세 크로마키 처리 과정은 다음과 같다. 먼저 크로마키 처리에 필요한 이미지 촬영이 필요하다([Fig. 3](b) 참조). 모바일 디바이스의 촬영 모드는 한 손 촬영이 가능한 세로 모드(portrait)로 고정하였다. 때문에 원본 이미지의 종횡비를 유지하여 촬영하도록 세로 모드에서 촬영한 이미지를 클리핑(clipping) 처리하고 있다.

다음으로 색상, 명도, 채도 값 조정을 통해 원하는 배경색상을 HSV 공간에서 크로마키 처리하도록 배경 색상, 채도, 명도를 조절하도록 구현하였다([Fig.3](c) 참조). 실시간으로 색상 변경이 가능하여 사용자는 원하는 수준의 전경 객체를 분리할 수 있다.



(a) Original Page

(b) Photo Preview

(c) Filtered Preview

(d) Manual Editing

(e) Final Composed Page

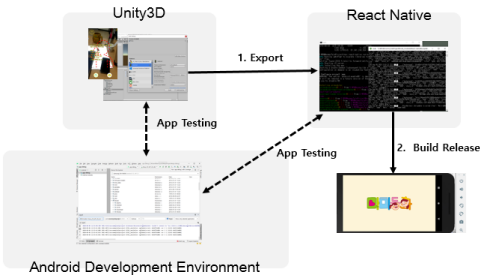
[Fig 3]. Chromakey Editing Process

하지만 크로마키 촬영 모드에서는 모든 전경 객체를 분리하지는 못하기 때문에, 분리되지 않은 부위를 분리하기 위해서 촬영 후에 사용자 편집 모드로 진입하여 원하는 부위를 지우거나 복원 한다([Fig.3](d) 참조). 원하는 부위를 지우고자 할 때

는 지우개 버튼을 눌러 손가락으로 원하는 부위를 드래깅하면 해당 부분이 투명 처리되어 픽셀들이 지워진다. 잘못 지워진 부위를 원상 복구시킬 때는 복원 버튼을 누르고 복원하고자 하는 위치를 손가락으로 드래깅하면 원본 이미지 픽셀로 복원된다.

4.2 개발 환경 연동

전체 페이지 구성 및 페이지 이동은 React Native로 개발되었으며, 크로마키 처리 및 이미지 편집 작업은 Unity3D로 개발되었기 때문에 두 개의 상이한 개발 환경에서 개발된 모듈들을 연동하는 기술이 필요하다. 본 연구에서는 통합 환경 구축을 위해 Beaulieu가 제시한 방식을 적용하였다[13]. [Fig.4]는 Beaulieu가 제시한 방식을 안드로이드 앱 개발 사례를 중심으로 도식화한 것이다.



[Fig. 4] Android App Building Process of Our Integrated Development Environment.

개별 개발 도구(Unity3D, React Native)들이 안드로이드 개발 환경과 연동을 통해 앱에 필요한 컴포넌트를 개발하고 테스트하는 과정은 점선으로 표시되어 있다. 컴포넌트 개발은 도구별로 독립적으로 진행된다. 실선으로 표현된 빌드 과정은 Unity3D 플랫폼에서 개발된 개발 결과물을 React Native에서 인식할 수 있도록 네이티브 모듈 형태로 변환하는 export 과정과 React Native에서 네이티브 모듈과 상호 통신하는 연동 코드 작성을 통해 개발된 모바일 앱 결과물을 빌드하고 추출하는 build release 과정으로 구성되어 있다. React Native에서는 Unity3D로 만들어진 네이티브 모듈을 인식할 수 있도록 React Native 컴포넌트 형태로 제작된 UnityView를 사용한다[14].

React Native로 개발된 컴포넌트와 Unity3D로 개발된 컴포넌트는 가이드 URI 이미지 정보(종횡비 정보, URI 주소 정보 등) 전달과 편집 완료 정보 등을 지속적으로 교환할 수 있어야 하며, 이를 위해 UnityView에서 제공하는 메시지 전달방식을 사용한다. 이러한 메시지 정보들을 JSON 형태로 가공하여 전달하도록 구현하였다.

이러한 통합형 개발 연동 방식에 대한 장단점을 요약하면 [Table. 1]과 같다. 본 연구에서 제안하는 통합 개발 환경은 웹에서 사용하는 CSS 같은 기술을 적용함으로써 UI 설계 및 개발 프로세스를 획기적으로 단축할 수 있으며 UI 관련 유지보수를 수월하게 해준다.

[Table 1] Pros and Cons of Our Integrated Development Environment

Pros	Quicker UI/UX Component Updates
	Easier Maintenance
Cons	Frequent Updates on the Platforms
	Rebuild Upon Every Component Update

반면, 다음과 같은 통합 개발 환경의 단점들도 있다. 첫째, React Native와 Unity3D 게임엔진의 버전에 따라서 통합 개발 환경을 구축할 때 추가 혹은 삭제 방식이 상이하다. React Native는 지속적으로 업데이트되는 오픈소스로 기능의 추가, 변경, 삭제가 빈번하다. 그래서 통합 개발 환경 구축 시 특정 React Native 버전과 특정 Unity3D 게임엔진 버전이 충돌하여 연동 오류가 발생할 수 있다. 둘째, Unity3D 게임엔진에 기능의 추가, 변경, 삭제할 때마다 다시 빌드하여 변환해야 한다.

4.3 캐싱 기반 이미지 관리 최적화

웹서버에 등록된 이미지는 원본 소스 이미지를 저장하기 때문에 React Native로 개발된 모바일 앱 환경에서는 다수의 이미지 처리에 따른 메모리 사용량이 증가한다. 이에 따라 잦은 모바일 앱 다운 현상이 발생하였으며, 메모리 사용량 증가에 따른 모바일 앱의 불안정이 발생하였다.

이 문제를 해결하기 위해 React Native 시스템에서 제공하는 이미지 캐싱 기법을 적용하였으나, 많은 동화 페이지 다운로드 및 재생에 따라 캐싱 효과가 반감되는 현상이 발생하였다. 또한 캐싱 모델은 기존 이미지에서 제공하는 터치 조작 등의 기능과 연동되지 않는 경우도 발생하였다.

그래서 동화책 내 모든 이미지들은 UUID 형태로 식별 가능하게 구분한 후 URI를 부여하였다. 서버에 업로드된 이미지는 성능 향상 및 안정성을 높이기 위해 이미지 해상도를 낮춘 이미지로 변경하였으며, 모바일 앱은 서버로부터 해상도가 변경된 이미지를 받아 앱 내부 캐시에 저장하였다. 이미지를 읽을 때 캐시에서 먼저 이미지를 읽도록 URI 정보를 수정하였다. 따라서 캐시에 있으면 캐시에서 읽고, 없으면 서버에서 읽어오도록 함으로써 React Native에서 기존에 제공하는 Image 컴포넌트의 기능을 수정 없이 사용할 수 있었다.

5. 성능평가

본 장에서는 클라이언트에서 서버로 크로마키 처리된 이미지를 전송하는 데 걸리는 응답시간과 처리 성공률에 대한 성능평가 결과를 제시한다.

테스트에 사용한 웹서버는 카페24 웹 호스팅에서 제공하는 10G 광아우토반 full SSD 퍼스트클래스 6G 서비스로서 하드디스크 용량은 4GB, 스트리밍 용량은 5.5GB로 PHP 7.3 MariaDB 10.0 버전이 설치되어 있다. 테스트로 사용한 모바일 디바이스들은 [Table 2]와 같다.

모바일 앱 개발에 Unity3D는 2019.2.17.f1 버전, React Native는 0.62 버전을 사용하여 개발하여 통합하였으며, 2020년 3월 24일부터 5월 4일에 걸쳐 운용성 및 성능 테스트를 진행하였다.

[Table 2] Android Devices Used for Performance Measurements

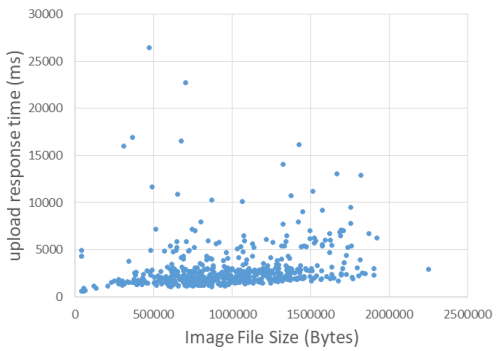
Android Version	Model Name
10.0	Galaxy S10e, S20, Note10+
9.0	V20, Note9
7.0	A8, A7
6.0	Galaxy S5, Note4, A5
5.0	Galaxy Note3

Android Version 8.0의 경우, LG V20으로 테스트 진행하였으나, 앱 기동시 동작하는 스플래시 화면과 설계 불일치 문제가 발생하였다. 버전 9.0에서 이 문제를 수정하였기 때문에 8.0 버전이 설치된 모바일 앱은 모두 9.0으로 업그레이드를 한 후에 운용성을 확인하였다. 테스트에 사용한 기기 11개(삼성 10종, LG 1개) 모두 모바일 앱 운용성 테스트를 통해 로그인, 로그아웃, 동화 목록 보기, 동화 재생, 동화 페이지 편집 기능들이 정상 동작함을 확인하였다.

특이 사항으로 A7기기 1종에 대해서만 해당 기기의 그래픽 하드웨어에서 크로마키 쉐이더가 정상적으로 동작하지 않았다. 실시간 크로마키 처리에는 문제가 없었으나 편집 시에 문제가 발생하였다. 소프트웨어적으로 보정한 버전을 개발하여 이 문제를 해결하였으나, 다른 기기에 비해 크로마키 편집 속도가 늦었다. 동일 회사 제품군에도 기능상 다양한 문제가 발생하여 안드로이드 기기에서의 운영은 여전히 파편화 현상이 있음을 확인하였다.

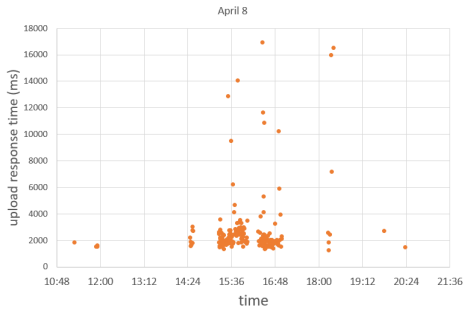
크로마키 처리된 이미지의 데이터 업로드 속도는 이미지 파일의 크기에 의존한다. 크로마키 처리되지 않은 일반적인 압축된 이미지 파일의 경우, 촬영 해상도에 크게 좌우되며 본 연구에서 사용한 1080x1920 이미지 해상도를 기준으로 평균 2MB 정도의 용량을 차지하였다. 따라서 크로마키 처리에 따른 이미지 파일 용량 줄이기가 응답시간 성능에 큰 영향을 미친다. 테스트 기간 중 크로마키 처리로 생성된 모든 이미지에 대한 업로드 응답시

간을 측정하였다. 그 결과는 [Fig. 5]에 제시되어 있다. 그림에서는 크로마키 이미지의 추가 처리 작업을 통해 추출된 이미지 파일의 업로드 응답시간을 크로마키 이미지 파일의 크기별로 분포를 도식화하고 있다. 응답시간은 평균 2.962초(±2.566)이며 평균 이미지 파일크기는 979KB (±391KB)이다. 이상치(outlier)값에 따른 민감도를 줄인 중간 값 응답시간은 2.214초로 목표 수치인 3초 이내로 안정적이었으며, 파일크기 중간 값은 954KB로 이상치의 영향이 크지 않음을 알 수 있다.



[Fig. 5] Distributions of Upload Response Times over Different Image File Sizes

[Fig. 6]은 테스트가 가장 활발하게 진행되었던 4월 8일의 이미지 파일 업로드 응답시간(ms) 분포를 시간대별로 도식화하고 있다. 해당일 발생한 187건의 업로드 이벤트에 대한 평균 응답시간은 2.782초이었으며, 중간 값 응답시간은 2.036초였다. 그림에서 보는 바와 같이 테스트 중 업로드 성능이 극단적으로 저하되는 스파이크(spike)가 드물게 발생하였으나 지속성은 없었다. 테스트 중에 모든 모바일앱 상에서 프로파일링을 진행하였으며, 특이 현상이 발견되지 않았기 때문에 서버 혹은 무선망에 성능 과부하에 따른 일시적인 성능저하 현상으로 추정된다.



[Fig. 6] Distribution of Upload Response Times over Test Periods

6. 결 론

본 연구에서는 React Native 프레임워크와 Unity3D 게임엔진을 통합한 개발환경을 활용하여 각 크로스 플랫폼에 특화된 기술을 적절하게 적용하여 크로마키 처리를 기반으로 이미지를 합성하는 모바일 앱 개발 사례를 소개하였다.

UI/UX 요소는 React Native의 JSX를 기반으로 화면 구성 작업 및 상호작용을 단순화하였으며, 사용자 터치를 기반으로 실시간 이미지 촬영, 크로마키 처리 및 편집 기능 등은 Unity3D 개발 환경을 활용하였다. 이들 상이한 개발 환경을 통합 운영함으로써 각 플랫폼에서 제공하는 특화된 기능을 적절하게 분배하고 사용하여 7개월의 짧은 개발 기간에 2명의 개발 인원(UI/UX 디자이너 제외)으로 목표하던 기능과 성능의 요구사항들을 만족할 수 있었다.

본 연구에서 크로스 플랫폼들을 통합한 개발 환경 방식이 향후 모바일뿐만 아니라 PC 앱을 개발할 때 개발 비용을 절감하고 개발 시간을 단축할 수 있음을 보여주었다. 본 연구에서 소개한 통합 개발 방법이 확산되고 보편화되기를 기대한다.

ACKNOWLEDGMENTS

This work was supported by 2018 Hongik University Research Fund.

REFERENCES

[1] YoungHyun Chang and SangYeob Oh, "A study on the development of one source multi use cross-platform based on zero coding", Multimedia Tools and Applications Vol. 74, No. 7, pp. 2219-2235, 2015.

[2] Andreas Biørn-Hansen, Tim A. Majchrzak, and Tor-Morten Grønli, "Progressive Web Apps: The Possible Web-native Unifier for Mobile Development", In WEBIST, pp. 344-351, 2017.

[3] Majchrzak, T. A., Biørn-Hansen, A., and Grønli, T.-M, "Comprehensive analysis of innovative crossplatform app development frameworks", In Proc. 49th HICSS. IEEE Computer Society, pp. 6162-6171, 2017.

[4] https://unity.com

[5] https://www.unrealengine.com

[6] Scott Forstall and Imran Chaudhri, "Webview applications", U.S. Patent Application No. 11/145,560, 2006.

[7] B. Eisenman, "Learning react native: building native mobile apps with javascript", O'Reilly Media Inc., 2015.

[8] T. Norton, "Learning C# by developing games with unity 3D", Packt Publishing Ltd, 2013.

[9] A. Smith, J. Blinn, "Blue Screen Matting", Proc. of SIGGRAPH '96. pp. 259-268. 1996.

[10] Swarnendu Ghosh, Nibaran Das, Ishita Das, and Ujjwal Maulik, "Understanding Deep Learning Techniques for Image Segmentation", ACM Computing Surveys, Vol. 52, No. 4, Article 73, 2019.

[11] R. Battle and E. Benson, "Bridging the semantic Web and Web 2.0 with representational state transfer (REST)" Journal of Web Semantics 6(1), pp. 61-69, 2008.

[12] Juriy Zaytsev, Stefan Kienzle, and Andrea Bogazzi. "Fabric. js - a powerful and simple Javascript HTML5 canvas library", 2008.

[13] F. Beaulieu, "Part 1. Show Unity3D view in React-Native application. Yes it's possible!", URL: https://medium.com/@beaulieufrancois/show-unity3d-view-in-react-native-application-yes-its-possible-852923389f2d, 2018.

[14] React Native UnityView, "react-native-unity-view",URL:https://www.npmjs.com/package/react-native-unity-view, 2018.

김 승 준 (Kim, Seung Jun)

약 력 : 2012-2019 홍익대학교 게임소프트웨어전공 학사
2019-현재 홍익대학교 대학원 게임학과 석사과정

관심분야 : 크로스 플랫폼, 딥러닝, 캐릭터 애니메이션

서 범 주 (Seo, Beom Joo)

약 력 : 1996-2001 LG전자 DTV연구소 주임연구원
2009-2012 싱가폴국립대 Senior Research Fellow
2013-현재 홍익대학교 게임학부 조교수

관심분야 : 교육용 게임, 가상현실, 분산 멀티미디어 DBMS

조 성 현 (Cho, Sung Hyun)

약 력 : 1974-1978 서울대학교 계산통계학과 이학사
1989-1995 UCLA 컴퓨터과학과 이학박사
1996년-현재 홍익대학교 게임학부 교수

관심분야 : 게임 프로그래밍, 게임 인공지능