math_2020_080

Міністерство освіти і науки України

Чернівецький національний університет

імені Юрія Федьковича

Факультет математики та інформатики
(повна назва інституту/факультету)

Кафедра математичного моделювання
(повна назва кафедри)

Розробка мобільного додатку для візуалізації дизайнерських рішень за допомогою AR-технологій

Дипломна робота

Рівень вищої освіти - другий (магістерський)

Виконав:
студент 6 курсу, групи 607
спеціальності 124 – Системний аналіз
(назва спеціальності)
Дворський Денис Вікторович
(прізвище, ім'я та по-батькові)
Керівник к.ф.-м.н., доцент Дорошенко І.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

До захисту допущено:

Протокол засідання кафедри № 7

від „ 15 ” грудня 2020 р.

зав. кафедри _____ проф. Черевко І.М.

Чернівці – 2020

Анотація

В даній магістерській роботі розглянуто питання візуалізації та обробки інформації, зокрема вдосконалення візуалізації тривимірних об'єктів засобами технології доповненої реальності. Самостійною частиною є розробка проекту доповненої реальності.

ЗМІСТ

ВСТУП

Еволюційно склалось так, що людство завжди намагалось покращити свої умови проживання, паралельно з цим, намагаючись розширювати свої межі світосприйняття та можливості. Саме це можна назвати причиною технологічного “буму” XX століття. Саме це століття стало найбільш знаковим для прогресу, самореалізації та утвердження чогось нового, що дало великий поштовх для XXI століття. То що ж на рахунок нашого часу?

В даний час стають все більш актуальними такі поняття, як інновації, інноваційний розвиток, інноваційна сфера і багато інших понять, пов'язані з інноваційними процесами. Інноваційність міцно увійшла в наше життя і стала визначальною в розвитку всіх сфер життєдіяльності людини: в політиці, в бізнесі, в сфері освіти, медицини, військової промисловості. Ні в один з попередніх періодів розвитку людства інновацій не приділялося такої пильної уваги(тому що зазвичай щось нове було доступне лиш елітарним структурам), і ніколи раніше не було такої кількості інноваційних програм, інноваційних розробок, технічних рішень різного рівня.

Глобалізація віртуальної реальності привела до введення в науковий обіг нового терміну «доповнена реальність». Якщо поточні технології призначених для користувача інтерфейсів сфокусовані в основному на взаємодії людини і комп'ютера, то доповнена реальність за допомогою комп'ютерних технологій пропонує вдосконалення інтерфейсу людини і реального навколишнього світу.

Отож даною роботою хотів би донести розуміння того що дана технологія є набагато ближчою ніж ми думаємо, а що найголовніше, покажу як саме її можна вдало використати та зробити це візуально привабливо та доступно.

РОЗДІЛ 1. Доповнена реальність

1.1. Суть AR-технології

Доповнена реальність (англ. Augmented Reality, надалі AR) - це технологія, яка одночасно дуже проста, але в той же час багато для кого не зрозуміла. З назви ми можемо зрозуміти як саме буде реалізована технологія. А саме, ми отримуємо звичайну нашу реальність, але з деякими елементами, які не існують насправді, а бачити ми їх можемо і частково навіть взаємодіяти. Зазвичай, це реалізується за рахунок виведення якогось змодельованого предмета поверх реального світу на якомусь екрані. При якісному і повному застосуванні доповненої реальності, грань між проекцією, який доповнює наш світ, і реальністю може бути і зовсім непомітною.

У прикладному плані основним завданням доповненої реальності є не відділення кінцевого користувача від реального світу і занурення в якесь віртуальне оточення, а створення майданчика для інтерактивної взаємодії з об'єктом, який цікавить. У зв'язку з цим одним з головних переваг технологій доповненої реальності є те, що за допомогою комп'ютерної бази можна виробляти взаємодію з якимось фізичним об’єктом чином в режимі реального часу. Ці технології здатні зробити сприйняття інформації людиною набагато простішим і наочнішим. Необхідні запити будуть автоматично доставлятися користувачеві. За допомогою цих технологій реальні об'єкти набувають нових якостей і відкриваються з іншого боку для користувача.

У вузькому сенсі доповнену реальність розуміють, як процес об'єднання об'єктів з реального світу з об'єктами з віртуального, згенерованих комп'ютером. У нашому столітті з'являється незліченна кількість вражаючих гаджетів, пристроїв і додатків, які змушують

подумати, чи будуть предмети, речі та явища, які ми знаємо в їх теперішньому вигляді, завжди виглядати так?

Дослідник Рональд Азума (Ronald Azuma) виділив три ознаки, якими має володіти розширена реальність [1]. В першу чергу це комбінування реального та віртуального світу. Так само, важлива інтерактивність та тривимірне уявлення об'єктів. Існують деякі основні характеристики та вимоги, необхідні для нормального функціонування технологій доповненої реальності. Перш за все, в тому чи іншому вигляді необхідна наявність обчислювальної платформи, здатної створити умови для взаємодії з фізичним об'єктом. Іншим важливим елементом систем доповненої реальності є дисплей для відображення об'єктів доповненої реальності. Якщо раніше роль таких елементів виконували звичні для сприйняття монітори, то сьогодні їх роль виконують складні фізичні системи та різного роду спеціальні сенсори.

Наступною важливою умовою функціонування технологій доповненої реальності є можливість інтерактивного введення, зміни існуючих віртуальних умов, можливість інтерпретації досліджуваного об'єкта саме в тому вигляді, в якому це потрібно. При цьому система повинна бути зрозумілою для того, щоб нею в різних формах могли користуватися не тільки професіонали, але і широкий загал людей.

У сучасних умовах можна відзначити ще одна важлива вимога для ефективної роботи доповненої реальності - можливість зберігання та обміну даними. Щоб система або конкретна технологія могла проектувати об'єкт - вона повинна отримувати інформацію про цей об'єкт з будь-якого джерела. Сьогодні замість ручного введення умов і параметрів використовується інтернет, що дозволяє швидко отримувати величезну масу інформації по потрібному об'єкту та виконувати інтерактивний обмін з кінцевим користувачем. З розвитком широкого доступу до інтернету з'явилася можливість використовувати технології

віртуальної і доповненої реальності для створення штучного інтелекту для виконання складних завдань. Але поки що такі системи призначені для вивчення вузькому колу професіоналів, вони знаходяться на стадії прототипів, але в перспективі можуть бути широко застосовані.

На сьогоднішній день більшість досліджень в області доповненої реальності сконцентровано на використанні живого або інтерактивного відео, підданого цифровій обробці, «доповненого» комп'ютерною графікою. Більш серйозні дослідження включають відстеження руху реальних об'єктів, розпізнавання координатних міток за допомогою машинного зору та конструювання керованого оточення. Також існує, таке поняття, як віртуальна реальність, але слід розуміти, що це абсолютно інший напрямок. Доповнена реальність вносить коректування в сприйняття дійсності. У той час як віртуальна реальність, це повністю відокремлений створений світ.

Система доповненої реальності є посередником між реальністю і людиною, а значить, на виході вона повинна створювати сигнал для одного з таких органів. За типом подання інформації можна виділити наступні системи:

1. Візуальні системи, в основі яких лежить зорове сприйняття користувача. Такі системи створюють зображення, яке використовується людиною для досягнення його цілей.
2. Існують аудіосистеми, орієнтовані на слухове сприйняття. Ці системи можуть використовуватися в якості навігаторів. Коли людина досягає конкретного місця, заданого програмою, вони видають різні звуки. Можливе використання стереоскопічного ефекту, який орієнтує людину в просторі і допомагає досягти заданого об'єкта.

3. Аудіовізуальні системи являють собою комбінацію двох попередніх типів, однак аудіоінформація в них носить допоміжний характер, що доповнює візуальні функції.

Системам доповненої реальності повинен бути відкритий доступ до отримання інформації навколишнього середовища, так як віртуальні об'єкти як раз будуються на основі цієї інформації. Кожна з таких систем має певний набір пристроїв, за допомогою сенсорів, що дозволяють сприймати різні сигнали навколишнього середовища: звукові і електромагнітні імпульси, переміщення в просторі, прискорення тощо. Сенсори поділяються за їх призначенням, так як подібні за своєю природою сигнали можуть нести різну інформацію.

Крім вищевказаних систем, можна виділити системи за типом наступних сенсорів: геопозиційному системи, які орієнтуються на сигнали систем позиціонування GPS. Додатково до приймачів таких сигналів, ці системи обладнані компасом і акселерометром для визначення кута повороту щодо вертикалі та азимута.

Наступний вид систем - оптичні, що працюють із зображенням, яке гаджет отримує на одну або кількох камер. Камери можуть переміщатися разом з системою або незалежно від неї.

Крім перерахованого, системи різняться за рівнем взаємодії з користувачем. У ряді основних систем користувач грає пасивну роль, коли виступає спостерігачем за реакцією системи на зміну навколишнього середовища; а також виділяються системи, які вимагають втручання користувача - він як управляє роботою самої системи з метою досягнення результатів, так і змінює віртуальні об'єкти. За даним критерієм виділяються такі системи.

Перш за все, це автономні, які не потребують втручання користувача в своїй роботі. Ці системи надають інформацію про об'єкти: вони аналізують об'єкти, які знаходяться в полі зору людини і видають

довідкову інформацію, користувач розглядає картину в музеї та за допомогою програми доповненої реальності отримує додаткові дані про художника, про долю картини, історії зображеного сюжету тощо. Системи не передбачають взаємодію з користувачем і служать тільки для надання йому супровідних даних про об'єкт.

І нарешті, інтерактивні системи, що ґрунтуються на взаємодії з користувачем. На різні дії користувача такі системи дають різний відповідь. Подібні системи потребують пристрої введення інформації. В якості такого пристрою виступає якийсь екран сенсорного мобільного пристрою, планшет або спеціальний маніпулятор. Від специфіки системи залежить вибір системи пристрою. якщо користувачеві потрібно зробити якісь дії з віртуальним об'єктом, то досить вказівного пристрою. У разі імітації реальних процесів і виконання складних дій з об'єктами, використовуються спеціальні маніпулятори, які мають якесь різноманітне кількість ступенів свободи. Ця інтерактивність виражається в різному ступені свободи.

Існують також системи, які дозволяють користувачеві модифікувати віртуальне середовище. Це так звані системи-симулятори реального дії: використовуються в ситуації, коли користуватися реальними об'єктами немає можливості. Наприклад, існують віртуальні «примірочні», в яких користувач взаємодіє з інтерфейсом, щоб вибирати одяг з наявного набору і шляхом накладення шарів отримувати власні зображення в різних вбраннях.

Розрізняють і інші системи: в них користувачеві не потрібно змінювати віртуальне середовище. Він сам вибирає віртуальні об'єкти, які хоче бачити. Користувач також може цілеспрямовано впливати на віртуальні об'єкти, але не на рівні структури, а на рівні відображення. Користувач, таким чином, може робити, аффінні перетворення типу повороту, переміщення тощо. До таких груп відносяться різні

архітектурні системи, які дозволяють побачити, як впишеться в існуючу обстановку нова споруда або його частина, а також навігаційні і геоінформаційні системи. Подібні системи можуть, наприклад, показувати частини об'єктів інтересу, приховані іншими будівлями, інформацію про обраних об'єктах тощо.

Можна виділити системи додаткової реальності за ступенем мобільності. З одного боку, це стаціонарні системи, призначені для роботи в фіксованому місці; переміщення таких систем веде до часткової або повної припинення їх функціонування. З іншого боку, електронні пристрої, які можуть бути легко переміщені та часто їх переміщення і лежить в основі виконуваної ними функції.

Таким чином, вище були представлені різні підстави для класифікації різних типів систем доповненої реальності. Належність до того чи іншого типу визначається, перш за все, функціями системи, які наповнюють об'єкти якимись візуально новими властивостями, створюють можливість більш детального інформативного ознайомлення з об'єктом і дають можливість переміщатися індивіду (туристу, екскурсанту), наприклад, разом з транспортним засобом в міському середовищі, не створюючи додаткових витрат на її переміщення.

1.2. Історія походження

Ідея того, що на якомусь електронному дисплеї можна накласти дані на картинку з реального життя належить не якомусь інженеру, чи футуристу, а всього лиш письменнику який писав дитячі книги… Цим “пророком” був Френк Баум. Саме його збірка про чарівника з країни Оз стала популярною на весь світ. Про доповнену реальність, а саме про маркування людей яке видно лише через спеціальні окуляри, він писав ще у 1901 році у своїй книжці “The Master Key” [2].

Самому визначенню AR в цьому році виповнилось 30 років. Саме у 1990 році дослідник Том Кодел, співпрацюючи з авіабудівною компанією Boeing, вперше дав назву даному виду технологій [3].

Вже в 1992 році було реалізовано першу функціонуючу AR систему(Virtual Fixtures), яку розробив Лії Розенберг в лабораторії військово повітряних сил Сполучених Штатів [4]

Наступною важливою точкою розвитку даної технології стала презентація Новозеландських науковців портативного шолому AR, який в реальному часі міг добавляти якісь змодельовані об'єкти на екран який був прилаштований до шолома в 2004 році.[5] Особливість була якраз в тому, що до цього моменту даний процес міг реалізовуватись лише в приміщеннях, або проекції відбувались на якісь стаціонарні екрани, з цього технологія почала ставати максимально мобільною.

В 2015 році Microsoft анонсували Windows Holographic та HoloLens гарнітури. Дана подія супроводжувалася багатьма розмовами, тому що це мали стати першими пристроями які були не якогось професійного призначення, а для звичайних покупців. З цього моменту починається змагання за першість на полі доступності технології серед звичайних користувачів.

2016 року став роком коли про технологію доповненої реальності стали говорити як про таку, яка стає загальнодоступною. Підставами для таких слів стала шалена популярність гри Pokemon GO. І попри те, що творці цієї гри не були першими хто використав дану технологію, вони стали першими, хто зробили настільки масовий продукт.

Цікавим фактом буде те, що в 2017 році генеральний директор Apple inc., Тім Кук, сказав що доповнена реальність є наступною "великою ідеєю" після смартфонів, та вбачає що в найближчі 5-10 років ця технологія буде вкорінена у всі сфери нашого життя [6].

Логічним буде думка про те, що якщо доповнена реальність це проектування якогось зображення поверх картинки реального життя на якомусь екрані. З чого може виникнути питання, чи можна було б створювати проекцію напряму в око. Вбачаючи в цьому майбутнє компанія Samsung в 2019 році звернулась до управління по патентах і товарних знаках США зі своїм патентом лінз доповненої реальності. Зазначається, що південнокорейська компанія винайшла спеціалізовані контактні лінзи з вбудованою прихованою камерою

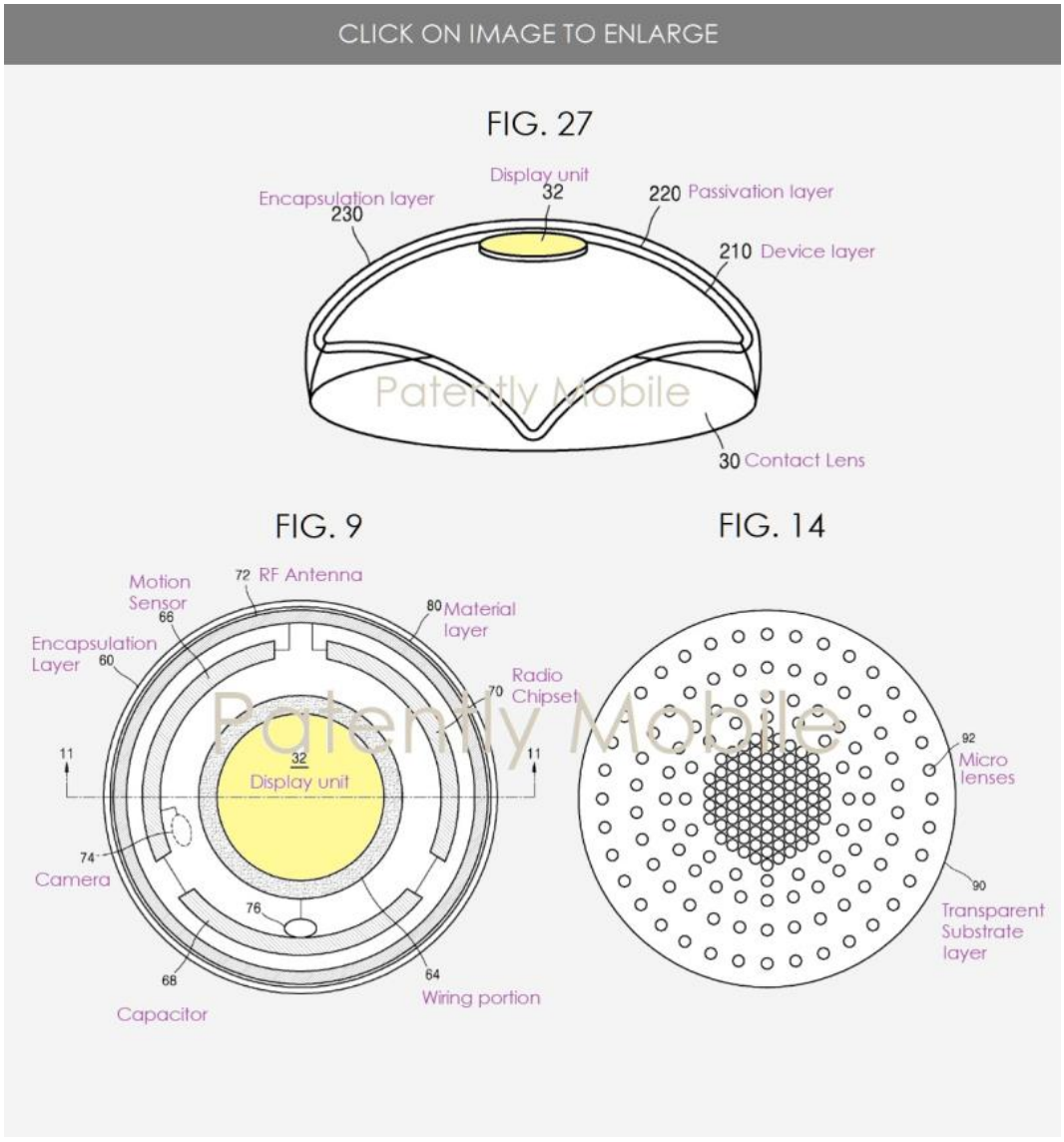


Рис.1.1. Патент спецлінз компанії Samsung

Те, що раніше вважалось науковою фантастикою, зараз цілком реалізоване та чекає на свої остаточні доповнення. Можливо саме ця технологія стане однією з головних в XXI столітті.

1.3. Сфери використання

Технологія AR може застосовуватися як для розваг і дозвілля, так і в професійній діяльності. Вона допомагає нам орієнтуватися в незнайомих місцях, а іноді навіть до невпізнання міняти нашу зовнішність.

Тому пройдемось коротко по тому де саме та як саме використовується технологія доповненої реальності:

- Медицина

  Поєднуючи телемедицину та технологію AR, асистування колег на складних операціях стає трішки легшим. Наприклад компанія Vipaar за допомогою Google Glass проектують руки колег на окуляри хірурга, що проводить операцію [8].

- Освіта

  Наразі все більшої популярності набирають заняття в яких навчання відбувається за допомогою AR. Але практика показує, що навчання таким чином можна проводити не лише для школярів, але й для професіоналів[9],[10].

- Авіація

  У військовій авіації вже досить давно використовується дана технологія, але для цивільної авіації дана технологія поки що достатньо складна в реалізації, але розвиток є і це головне [11].

- Туризм

  Дана сфера одна з перших почала достатньо широко використовувати технологію доповненої реальності. Для того

щоб “розвантажити” музеї, влада Роттердаму розробила екскурсію по найвизначніших деревах міста і за допомогою AR технології [12].

- Маркетинг

  Наразі наше суспільство позиціонується як споживацьке, тобто ми нарощуємо наш споживацький “апетит”. Маркетологи в свою чергу лише підігрівають дану ситуацію, але вражає як те як дана технологія дозволила неординарно підійти до того, щоб презентувати щось[13], [17].

- Дизайн

  Дуже цікавий напрямок розвитку AR технології, тому що часто люди візуально сприймають інформацію набагато краще. Даний напрямок допоможе більшості в тому щоб зрозуміти, якщо щось має виглядати [14]

- Покупки

  2020 рік став справжнім випробуванням для сфери послуг, але бізнеси, які змогли перенести свої процеси в Глобальну Павутину будуть мати змогу залишатись на плаву. Саме за допомогою AR онлайн магазини зможуть залучати ще більше покупці [15].

- Розваги

  Нікого не оминула лихоманка, яка накрила світ в 2016 році. Звичайно ж мається на увазі Pokemon GO. Хтось був безпосереднім учасником, а хтось лише дивувався будучи спостерігачем, як сотні людей в пошуках своїх “улюбленців”. [16]

1.4. Перспективи розвитку

Технології доповненої реальності в сучасних умовах не є чимось фантастичним. Але по-справжньому бурхливий розвиток за прогнозами буде пов'язано з появою великої кількості технологічно розвинених handsfree пристроїв. Це дозволить виконувати широкий спектр прикладних задач в різних сферах людської діяльності: навігація, складні медичні операції, складна картографія, нестандартні військові маневри. Одним з головних умов бурхливого розвитку handfree пристроїв є технологічний розвиток цифрової галузі в цілому. Однак вважається, що до піку розвитку сучасної електроніки ще далеко. Продукти, засновані на доповненій реальності, дозволять виконувати багато стандартних задач незвичайним способом та викликати у кінцевого споживача і користувача додаткову цікавість, залучаючи його до процесу.

Ще 10 років тому про існування доповненої реальності знали тільки фахівці, а всі спроби практичного використання можна було перерахувати по пальцях. Однак уже в 2010 році доповненою реальністю займалися десятки компаній, а в спеціалізованій пресі всерйоз обговорювалося питання про те, як скоро ця технологія перестане розбурхувати уяву, а стане чимось звичним і повсякденним, як для всіх став мобільний телефон. Всього лише за рік кількість додатків для iPhone, заснованих на використанні доповненої реальності в тій чи іншій формі, зросла з нуля до двох сотень. «Уже в кінці 2010 року в гру вступив сам Google з візуальним пошуковиком Google Goggles, а Nokia нарешті доробила давно обіцяне додаток Point & Find, призначене для розпізнавання штрих-кодів та кінопостери.

Переносна електроніка, яка так само може бути названа портативними комп'ютерними системами, - нова галузь, що розвиває комп'ютерні технології, в яких активно застосовуються технології віртуальної та доповненої реальності. Робота портативної електроніки

багато в чому пов'язана з роботою смартфонів та інших гаджетів. Тому постійне технологічне зростання смартфонів веде до зростання і портативної електроніки. Технологічні гіганти активно займаються розробкою даного виду електроніки. Слід зазначити, що сьогодні область портативної електроніки представлена кількома типами продуктів:

- "розумні окуляри" та інші схожі по застосуванню типи пристроїв, включаючи схожі системи віртуального доповнення реальності;
- спорт та фітнес-трекери, що дозволяють в реальному часі відстежувати фізичне навантаження;
- одяг з елементами електроніки, передає на смартфони дані про поточний стан організму;
- пристрої у форум-факторі годинників. У поєднанні зі смартфонами дозволяють дистанційно керувати різними діями, отримувати довідкову інформацію в зручному вигляді.

«Аналітики також прогнозують, що AR буде активніше використовуватися в соціальних медіа і навіть може «зруйнувати стіну між цифровим і фізичним простором». Тут мається на увазі те, що в реальному, фізичному світі з'являться цифрові аватари користувачів.

Ринок доповненої реальності буде рости там, де існує великий ринок сучасних смартфонів та планшетів, які є основною платформою для використання технологій віртуальної та доповненої реальності. Перш за все, до регіонів з такими ринками можна віднести Північну Америку, Західну Європу і Далекий Схід. Кращий потенціал в цьому плані має Японія.

Також зростання обсягів інвестицій пов'язаний не тільки з тим, що додатки доповненої реальності все активніше інтегруються в звичні для нас мобільні пристрої, а й з розвитком хмарних технологій, за допомогою яких користувач отримує миттєвий доступ до заздалегідь структурованої інформації. У технологіях доповненої реальності хмарні обчислення

використовуються для потокової обробки зображень (наприклад, розпізнавання зображень). Згідно з даними ABI Research, в недалекому майбутньому можливий розвиток вкрай складних систем, пов'язаних з Bigdata та колосальним об'ємом структурованої і неструктурованої інформації. Ймовірно, поява мережі "інтернету всього", в якому з мережею інтернет так чи інакше будуть пов'язані практично всі навколишні нас об'єкти.

З урахуванням цих тенденцій та зростання ринку переносної електроніки і смартфонів, Juniper Research прогнозує, що доповнена реальність стане однією з найважливіших платформ для комунікації і торгівлі. Екосистема додатків для доповненої реальності буде розширюватися. Основне зростання буде відбуватися за рахунок ігрової індустрії, але також у великих обсягах нові цифрові технології будуть застосовуватися в спорті, медицині, освіті, а в подальшому, для вирішення прикладних завдань в ЗМІ, рекламі тощо.

Прикладну значимість технології Augmented Reality складно переоцінити, тому що це єдина технологія, яка так стрімко пройшла шлях від комп'ютерних ігор, де вона виконувала розважальну функцію, до складних медичних апаратів, несучи в собі суспільну користь. Незважаючи на те, що ця область стала розширюватися і рости зовсім недавно, сьогодні це одна з найбільш перспективних сфер в області комп'ютерних технологій. Пов'язано це з повсюдним поширенням переносної електроніки, планшетів, смартфонів, за допомогою яких можливе використання цієї технології не тільки для наукових і технічних цілей, а й у повсякденному житті.

Дослідження від Juniper Research показує, що число користувацький додатків досягне 10 млрд. до 2024 року, порівняно з 3 млрд. на теперішній час. У своєму дослідженні “Consumer Mixed Reality: Emerging Opportunities, Vendor Strategies & Market Forecasts 2019-2024”

[7]визначає, що соціальні медіа та ігри є ключовим рушієм в збільшенні цифри користувацьких додатків і буде складати більше половини ринку до 2024 року.

Також згідно з даними, продаж від додатків, що використовують змішану реальність складе 8.57 млрд. фунтів стерлінгів, порівняно з 1.5 млрд. фунтів на 2019 року.

Також неможливо не згадати про останнє дослідження аналітичного центру Strategy Analytics, яке має назву ‘Short and Long-Term Impacts of Covid-19 on the AR and VR Market.’ в якому говориться про те що, навіть попри великий удар по доходах всіх компаній, даний вид технологій відповідно до того як компанії позиціонують свій продукт, може дуже збільшити загальний дохід.

- Дохід від технологій змішаної реальності складе 28 млрд. доларів США у 2025 році обто прогнозуєтсья, що прибутки виростуть у 6 разів.
- 2021 та 2022 роки будуть роками “повернення” смартфонів, які презентувались як VR гарнітура через поширення технології 5G.

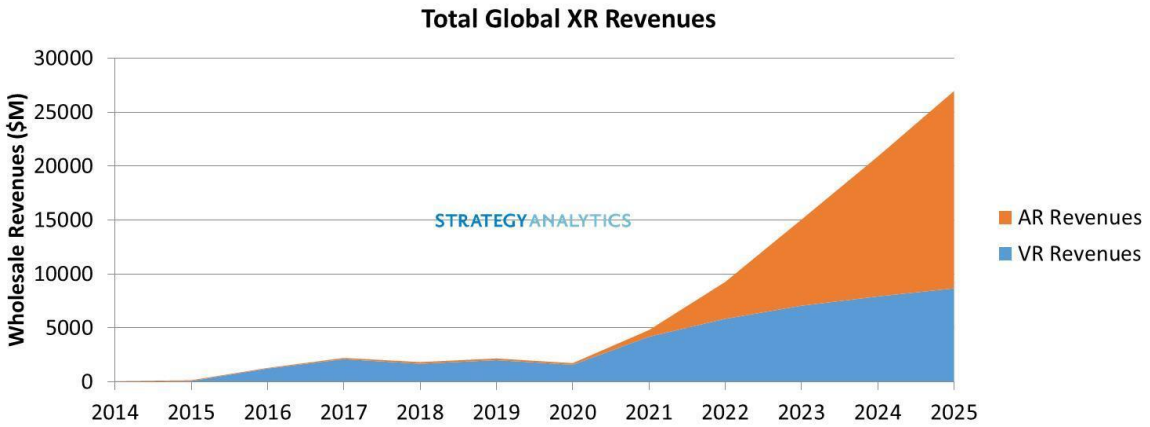


Рис.1.2. Оцінка прибутку від технологій змішаної розширеної реальності

РОЗДІЛ 2. Технології які дозволяють написати AR додаток для Apple

2.1. ARKit

ARKit – це програмний каркас для розробки програм доповненої реальності. З ним ви можете доповнювати віртуальним 2D та 3D контентом, базуючись на камері, якою ви знімаєте все навкруги. Цей каркас не є новий, наприклад програмному каркасу Vuforia вже багато років, проте ARKit визначає саме те, що використовує безмаркерний трекінг. Тобто йому не потрібно карток (здебільшого це QR коди чи щось подібне). Він розуміє світ навколо себе та швидко може визначити поверхні на які потім може розміщувати віртуальні об’єкти. Визначним є те, що старі програмні каркаси включили ARKit у свої API, що дало змогу їм надбати особливість безмаркерного трекінгу.

Отож що таке ARKit – це...

- high-level API. Це означає, що у вас буде зрозумілий інтерфейс для взаємодії.
- mobile AR platform дозволяє створювати віртуальні об'єкти та розміщувати їх в реальному світі.

З обмежень у нас:

- iOS 11 and up (A9 and up). ARKit підтримується пристроями iOS 11 і вище. Також процесор на цих пристроях повинен бути не гірше ніж A9.
- iPhone SE, iPhone 6s, iPad Pro, iPad (2017) and newer. Варто пам'ятати, що ARKit підтримується не всіма пристроями, і в цьому пункті наведені з яких починається підтримка ARKit. Більш нові пристрої будуть підтримувати ARKit без проблем.

ARKit складається з трьох шарів:

1. **Tracking** - це сенсори девайса, які відстежують зміни з самим девайсом.
2. **Scene understanding** - це розуміння того, що відображається через камеру. Наприклад, це визначення відстані до об'єкта або визначення освітленості.
3. **Rendering** - цей шар працює з даними отриманими в перших двох шарах і саме цей шар готовий відображати картинку.

Тепер детально зупинимося на кожному з шарів. І так Tracking базовий шар на якому будується ARKit. Цей шар дозволяє відстежувати позицію пристроїв в реальному часі. World tracking дозволяє відслідковувати не тільки орієнтацію нашого пристрою, але а також положення нашого пристрою щодо навколишнього середовища. Для більш точного визначення пристрою в реальному світі використовується метод візуальної одометрії (Visual Interial Odometry).

Одометрія - це метод оцінки стану і орієнтації пристрою за допомогою аналізу послідовності зображень, знятих встановленою на ньому камерою.

І один з вагомих плюсів ARKit це те, що не потрібно робити ніяких додаткових налаштувань.

Після Tracking настає черга наступного шару Scene understanding. Розуміння сцени, розуміння того, що відбувається через камеру девайса.

Одним з процесів цього шару є Plane detection. Що б розмістити об'єкт нам потрібна поверхня. І якраз тут в гру вступає Plane detection.

Після того як поверхня знайдена, необхідно отримати координату на цій поверхні, куди можна поставити віртуальний об'єкт. Для цього використовується ще одне процес – Hit-testing. У цьому процесі порівнюються координати реального світу і віртуального об'єкта. Потім ці координати поєднуються та в результаті виходить віртуальний об'єкт розташований в реальному світі.

Також на цьому шарі відбувається оцінка освітленості Light estimation. Робиться це для того, щоб віртуальний об'єкт виглядав гармонійно в реальному світі.

І останній шар це Rendering. Одним з властивостей цього шару є Easy integration. Це означає проста інтеграція в будь-який визуализатор. Так як використовується стрім фіксованого числа зображень, значить у нас є вся інформація для відтворення сцени. Отже ми можемо помістити ці дані в визуализатор і отримати картинку. Усередині ARKit є ARViews, які реалізують більшу частину роботи по рендерингу всередині себе.

Для тих хто працює з кастомними рендерингом всередині Xcode є окремий шаблон який працює з Metal і дозволяє інтегрувати ARKit в кастомний візуалізатор.

У ARKit під "капотом" є фреймворки AVFoundation і CoreMotion. AVFoundation необхідний для того, щоб можна було захоплювати зображення. CoreMotion отримує дані з сенсорів нашого пристрою.

Сам фреймворк ARKit має API засноване на ARSession. З цим API і відбувається основна робота під час розробки додатків доповненої реальності.

Сесії налаштовуються в ARConfiguration. А на виході з сесії виходять кадри або фрейми доповненої реальності ARFrame.

2.2. SwiftUI

До досконалості йому ("UI" розшифровується і перекладається як "призначений для користувача інтерфейс", значить це "він") ще розвиватися і розвиватися, але задуманий він настільки красиво і оригінально, що залишитися байдужим не виходить. Його рівняють з брудом (перераховуючи функції які є в XAML, Flutter і в інших аналогах, але немає в "новонародженого"). Його називають заміною AppKit (в macOS) і UIKit (в iOS і iOS-подібних системах), але це неправда. SwiftUI

це окремий програмний каркас і складова частина Xcode 11, він бере на себе частину функцій AppKit і UIKit, при цьому використовуючи функції AppKit і UIKit для, наприклад, відображення інтерфейсів і для їх взаємодії з користувачем. За словами одного з інженерів групи SwiftUI, це тимчасово: в майбутньому на місці AppKit або UIKit буде щось інше.

А ось життєвий шлях Interface Builder (IB) наближається до кінця - SwiftUI робить те ж що і IB, тільки цікавіше й ефективніше. У 1986 році доктор комп'ютерних наук Жан-Марі Юлло́ написав програму InterfaceBuilder (на LISP) для компанії ExperTelligence. А LISP це, "декларативний варіант Фортрана". Одного разу, в 1987, Жана-Марі привезли в офіс NeXT і познайомили з Джобсом. У 1988 році в NeXTSTEP 0.8, до складу якої входив IB написаний на Objective-C. IB використовувався при написанні першого в світі браузера WorldWideWeb. Останнє його оновлення сталося 9 років тому, в 2011 році. На WWDC 2011 року багато чого обіцяли в нього додати, то що показали в кулуарах виглядало менш фантастично ніж Canvas в SwiftUI, але для того часу це було б здорово - на жаль.

IB і до 2011 оновлювали нечасто, а з 2008 року число користувачів Xcode стрімко зростало, і майже 90% нових користувачів писали програми для iPhone OS (як тоді називалася iOS), а з них IB в своїй роботі використовував кожен третій. Так склалось. Писати для iPhone OS почали задовго (за півроку) до першої бета-версії iPhone SDK, і в програмах від Apple слідів IB (файлів з розширенням Xib) не було. Стів обіцяв що в SDK обов'язково буде IB, але змусити цей інструмент працювати з мобільної ОС виявилося важче, ніж перенести його з LISP в Objective-C. У перших бетах SDK його не було. Аж до iPhone SDK 2.0 він працював погано, а саме тоді люди освоювалися з новою для них системою, набиралися досвіду і набували звичок.

В результаті, призначені для користувача інтерфейси програмісти iOS і iOS-подібних систем пишуть або за допомогою IB, або у вихідному коді. Більшість користувачів відмічають, що SwiftUI дійсно зрозуміліший та менш вибиває помилок ніж IB. Для тих хто малює інтерфейси в початкових кодах SwiftUI це ще краще, тому що зараз для того, щоб розташувати на екрані елемент інтерфейсу, нехай це буде текстова рядок, будь-який напис, програміст повинен:

- створити об'єкт відповідного типу;
- обчислити координати задають положення об'єкта на екрані;
- розмістити об'єкт на екрані;
- задати значення об'єкта;
- задати атрибути тексту (це кілька рядків, як правило - шрифт, стиль, колір тощо)

Це один з найпростіших об'єктів - створення таблиці (списку) вручну може зайняти не одну сотню нетривіальних рядків. Вважається що в кожних ста рядках нового коду, як мінімум, повинні виявитися дві помилки, які не будуть виявлені при тестуванні і проявлять себе тільки потрапивши в руки користувача.

У SwiftUI використовується декларативна парадигма програмування, в рамках якої ця деталізація і всі ці тисячі рядків стають не потрібні. Програміст вказує що, і як щодо інших елементів, має бути виведено на екран (все це негайно відображається на екрані Canvas як буде виглядати в реальній програмі), а крім того, інженери Apple додали в цю і без того вражаючу картину сотні божевільних, але дуже корисних і зручних, трюків.

2.3. RealityKit

RealityKit – це програмний каркас для 3D-моделювання та рендерингу. RealityKit використовує інформацію, надану програмним каркасом ARKit, для плавної інтеграції віртуальних об’єктів у реальний світ.

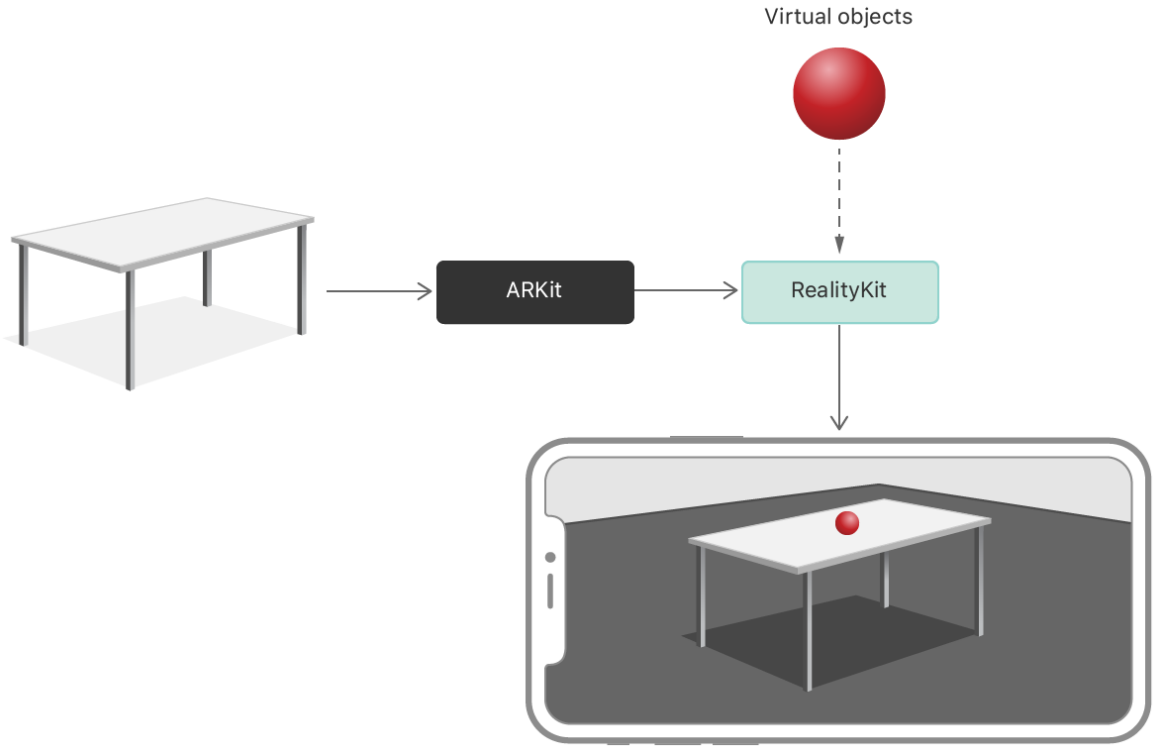


Рис 2.1. Послідовність використання програмних каркасів

Отож RealityKit має таку основну функціональність:

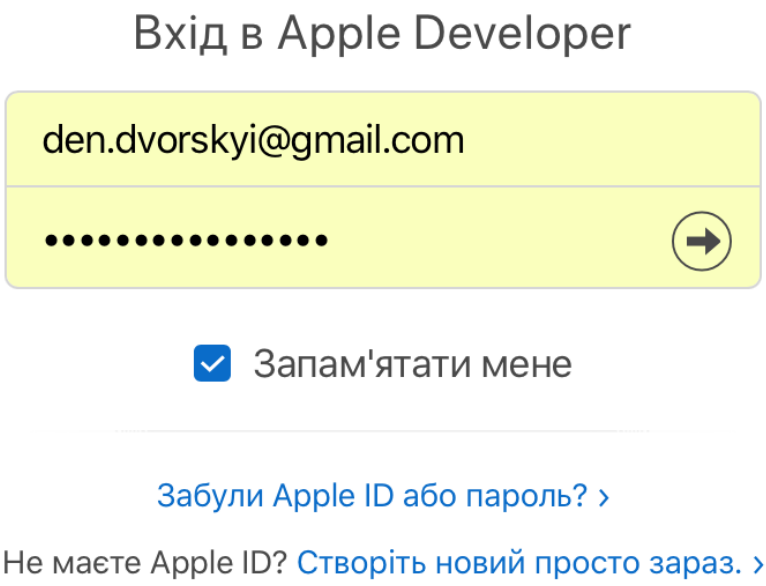
- імпортувати повністю сформовані заготовки, які ви можете зробити за допомогою програми Reality Composer, або створюйте їх із сіток, матеріалів та текстур;
- розміщувати джерела звуку в навколишньому середовищі;
- анімувати об’єкти як вручну, так і за допомогою фізичного моделювання;

- реагувати на взаємодію користувачів та зміни в навколишньому середовищі;
- синхронізувати дані між пристроями, забезпечуючи груповий досвід AR.

РОЗДІЛ 3. Розробка програмного продукту

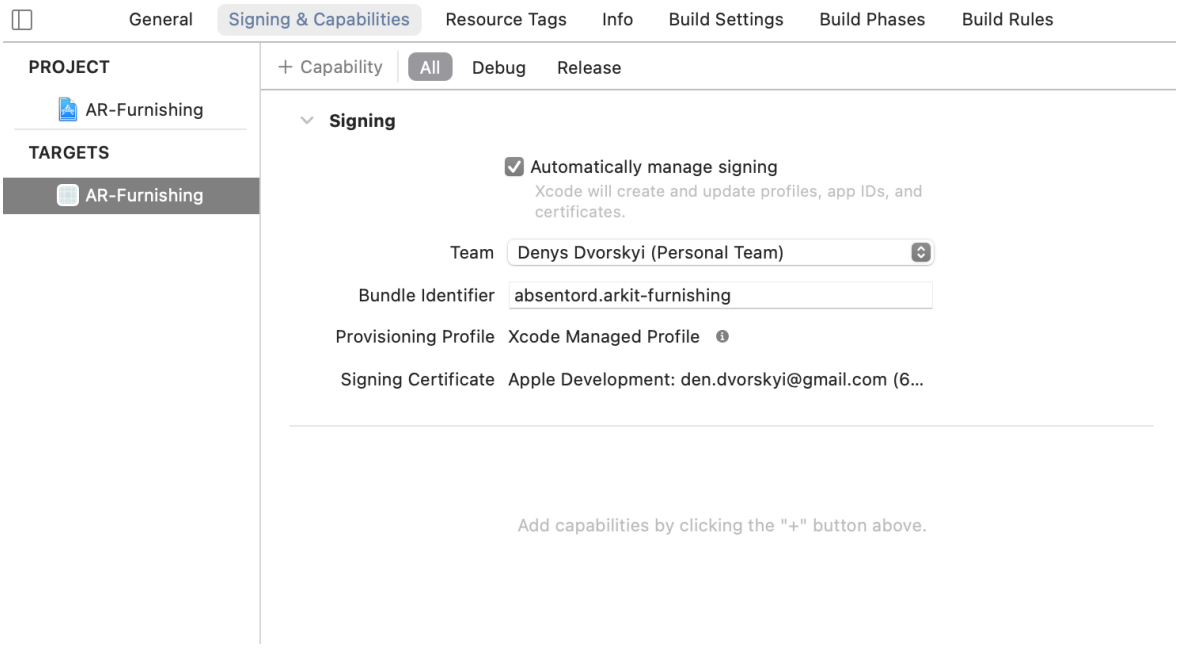
Оскільки продукція Apple є достатньо закритою, то для того, щоб змогти щось запрограмувати нам знадобиться Xcode 11+ та телефон з операційною системою iOS 11(або вище) та з процесором A9 (або вище). Тобто для цього нам підійдуть лише технічка Apple, оскільки на пристроях з іншими операційними системами програмування під iOS неможливі.

Для того, щоб програмувати під iOS зареєструвати себе на офіційному сайті https://developer.apple.com [18]

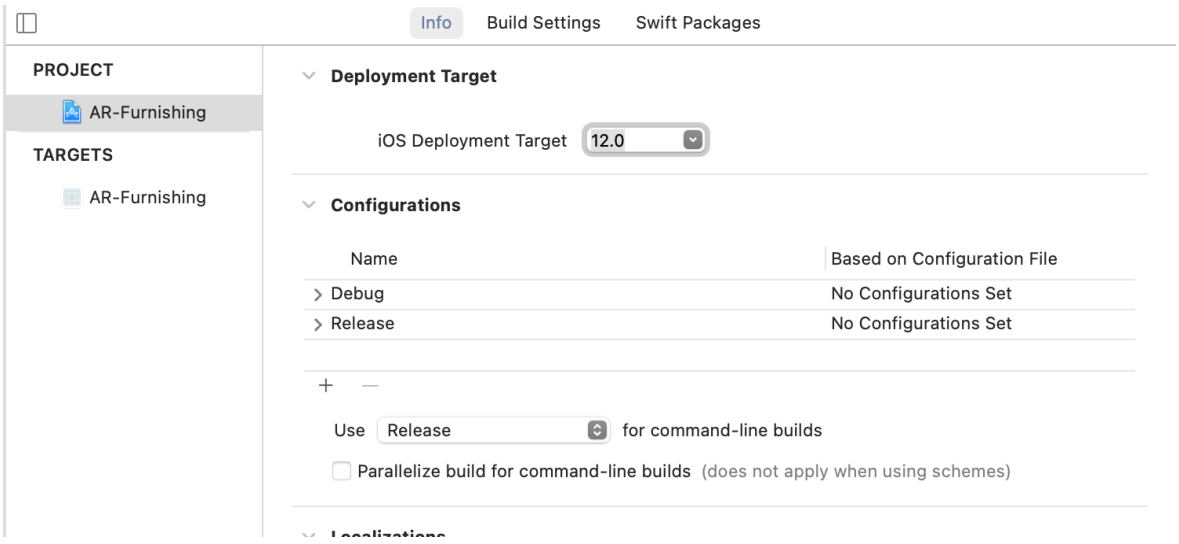


Дана маніпуляція робиться для того, щоб ми змогли “збудувати” додаток вибравши свою команду в рядку команди , тому що без цього ми не зможемо навіть подивитись чи воно буде працювати. В мене пише (Personal Team) тому що додатки зазвичай мають свої дозволи для своїх

команд.



Як і писалось вище дана технологія була доступна ще з iOS 11, але в даному проекті будуть використовуватись дещо що працює лише на після 12 версії операційної системи, тому розробку будемо вести для цих версій.



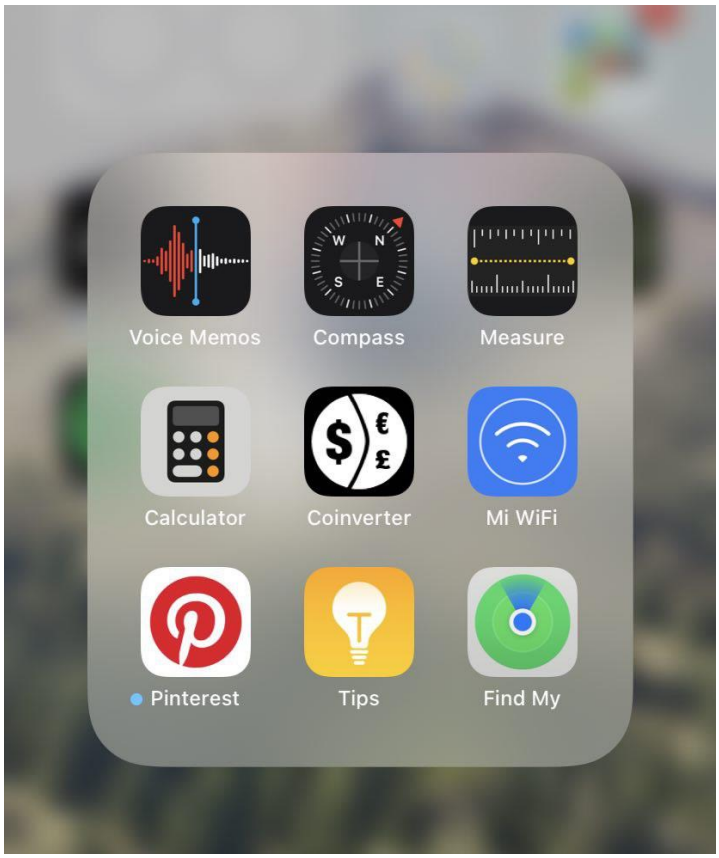
Спочатку створюємо “делегат”, який перевіряє чи програма запустилась правильно та після якого буде показуватись основне тіло програми://

```
import UIKit
import SwiftUI

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

   var window: UIWindow?

   func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

      // Create the SwiftUI view that provides the window contents.
      let contentView = ContentView()

      // Use a UIHostingController as window root view controller.
      let window = UIWindow(frame: UIScreen.main.bounds)
      window.rootViewController = UIHostingController(rootView: contentView)
      self.window = window
      window.makeKeyAndVisible()

      return true
   }

}
```

Після цього я хотів би “навчити” розуміти програму, щоб коли ми змінюємо орієнтацію телефона, програма та іконки також повертались разом з телефоном

```
import Foundation
import UIKit
import SwiftUI

class Device: ObservableObject {

   @Published var orientation: UIDeviceOrientation = UIDevice.current.orientation

   private var _observer: NSObjectProtocol?

   init() {
      _observer = NotificationCenter.default.addObserver(forName: UIDevice.orientationDidChangeNotification, object: nil, queue: nil) { _ in
         self.orientation = UIDevice.current.orientation
      }
   }

   deinit {
      if let observer = _observer {
         NotificationCenter.default.removeObserver(observer)
      }
   }

   func getImageOrientationAngle() -> Angle {
      switch orientation {
      case .landscapeLeft:
         return Angle(degrees: 90)
      case .landscapeRight:
         return Angle(degrees: -90)
      default:
         return Angle(degrees: 0)
      }
   }
}
```

Компанія apple має достатньо жорсткі гайдлайни по тому як мають виглядати їхні програми і багато вважає це великим недоліком, тому що так ніхто не може зробити дизайн під себе. Хоча я й займаюсь розробкою некомерційного продукту, всерівно вважаю що потрібно прагнути до уніфікації вигляду додатків і тому хочу використати розмиття яке ми часто бачимо на iPhone, а саме фон який ми бачимо одразу за іконками.




```
import SwiftUI

struct Blur: UIViewRepresentable {

    var style: UIBlurEffect.Style = .systemMaterial

    func makeUIView(context: Context) -> UIVisualEffectView {
        return UIVisualEffectView(effect: UIBlurEffect(style: style))
    }

    func updateUIView(_ uiView: UIVisualEffectView, context: Context) {
```

```
        uiView.effect = UIBlurEffect(style: style)
    }
}
```

Надалі ми мусимо наповнити моделями меблів, а також зробити так, щоб ми одразу бачили, що ми хочемо вибрати для розміщення.

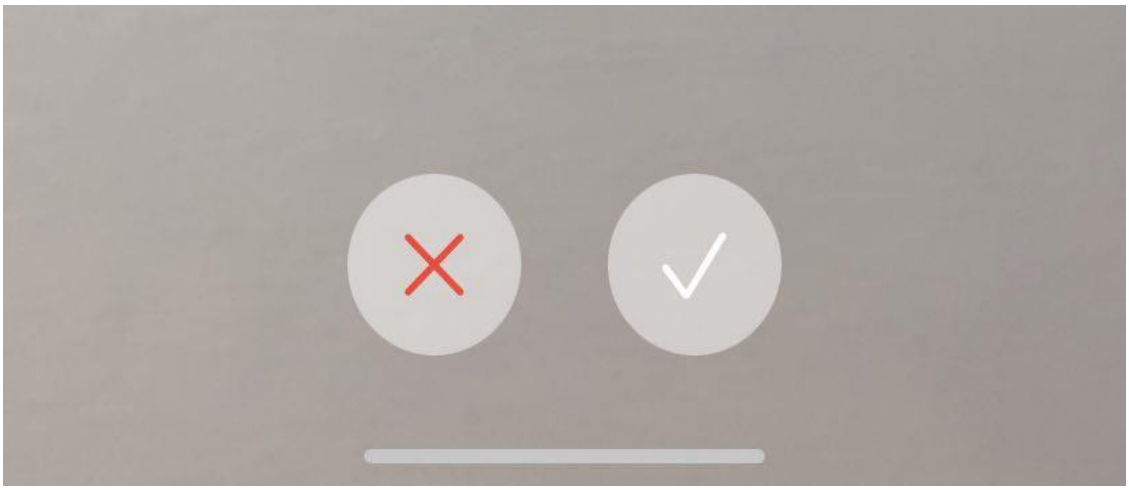
```
import SwiftUI

struct ContentView: View {

    @State private var modelPlacementEnabled: Bool = false
    @State private var selectedModel: Model?
    @State private var modelConfirmedForPlacement: Model?

    private var models: [Model] = {
        let filemanager = FileManager.default

        guard let path = Bundle.main.resourcePath, let files = try? filemanager.contentsOfDirectory(atPath: path)
        else { return [] }

        var availableModels: [Model] = []
        for filename in files where filename.hasSuffix("usdz") {
            let modelName = filename.replacingOccurrences(of: ".usdz", with: "")
            let model = Model(name: modelName)

            availableModels.append(model)
        }

        return availableModels
    }()

    var body: some View {
        NavigationView {
            ZStack(alignment: .bottom) {
                ARViewContainer(selectedModel: $selectedModel, modelConfirmedForPlacement: $modelConfirmedForPlacement).edgesIgnoringSafeArea(.all)
```

```
            if modelPlacementEnabled {
                PlacementButtonsView(modelPlacementEnabled:
$modelPlacementEnabled, selectedModel: $selectedModel,
modelConfirmedForPlacement: $modelConfirmedForPlacement)
            } else {
                ModelPickerView(modelPlacementEnabled:
$modelPlacementEnabled, selectedModel: $selectedModel, models: models)
            }
        }
    }
  }
}

struct ModelPickerView: View {
  @Binding var modelPlacementEnabled: Bool
  @Binding var selectedModel: Model?

  @ObservedObject var device = Device()

  var models: [Model]

  var body: some View {
    VStack {
      HStack {
        Spacer()
        NavigationLink(destination: ExportView()) {
          Text("Export")
            .font(.body)
            .foregroundColor(Color("primaryTextColor"))
            .padding(6)
        }
        .background(Blur(style: .regular))
        .cornerRadius(12)
      }

      ScrollView(.horizontal, showsIndicators: false) {
```

```
            HStack(spacing: 6) {
                ForEach(0 ..< models.count) { index in
                    Button(action: {
                        selectedModel = models[index]

                        modelPlacementEnabled = true
                    }) {
                        Image(uiImage: models[index].thumbnail)
                            .resizable()
                            .frame(height: 64)
                            .aspectRatio(1/1, contentMode: .fit)
                            .background(Color.white)
                            .cornerRadius(12)
                            .rotationEffect(device.getImageOrientationAngle())
                            .animation(Animation.easeInOut)
                    }
                    .buttonStyle(PlainButtonStyle())
                }
            }
            .padding(12)
        }
        .background(Blur(style: .regular))
        .cornerRadius(20)
    }
    .padding(6)
  }
}

struct PlacementButtonsView: View {
  @Binding var modelPlacementEnabled: Bool
  @Binding var selectedModel: Model?
  @Binding var modelConfirmedForPlacement: Model?

  var body: some View {
    HStack {
      Button(action: {
        resetPlacementState()
```

```
            }) {
                Image(systemName: "xmark")
                    .frame(width: 64, height: 64)
                    .font(.title)
                    .foregroundColor(Color.red)
                    .background(Color.white.opacity(0.5))
                    .cornerRadius(32)
                    .padding(12)
            }

            Button(action: {
                modelConfirmedForPlacement = selectedModel

                resetPlacementState()
            }) {
                Image(systemName: "checkmark")
                    .frame(width: 64, height: 64)
                    .font(.title)
                    .foregroundColor(Color.white)
                    .background(Color.white.opacity(0.5))
                    .cornerRadius(32)
                    .padding(12)
            }
        }
    }

    func resetPlacementState() {
        selectedModel = nil
        modelPlacementEnabled = false
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

Також в коді бачимо, що в нас є кнопки які дозволять або поставити якийсь об’єкт, або продовжити вибір предмету інтер’єру, які візуально виглядають наступним чином:

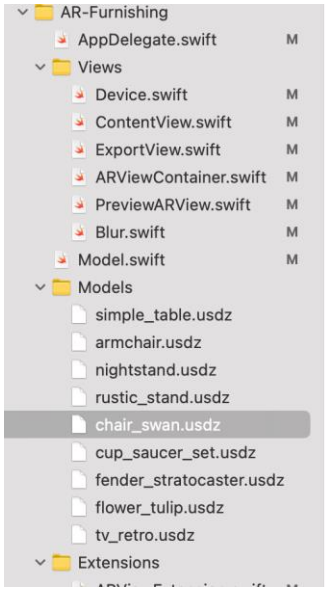


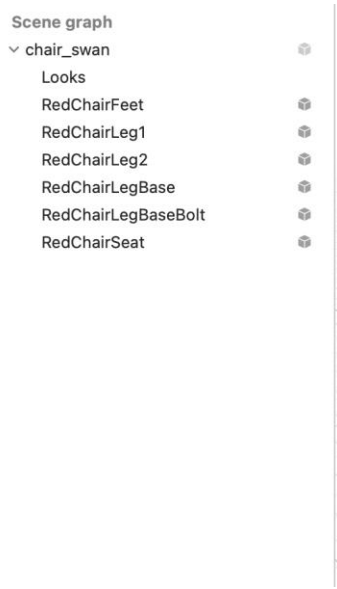
Наступне, нашій програмі потрібно розуміти які саме моделі маємо вибирати, та якось їх якось візуалізувати для користувача

```
import UIKit
import RealityKit
import Combine

class Model {
    var name: String
    var thumbnail: UIImage
    var entity: ModelEntity?

    private var cancellable: AnyCancellable? = nil

    init(name: String) {
        self.name = name
        self.thumbnail = UIImage(named: name) ?? UIImage(named: "placeholder")!

        let filename = name + ".usdz"
        self.cancellable = ModelEntity.loadModelAsync(named: filename)
            .sink(receiveCompletion: { loadCompletion in
```

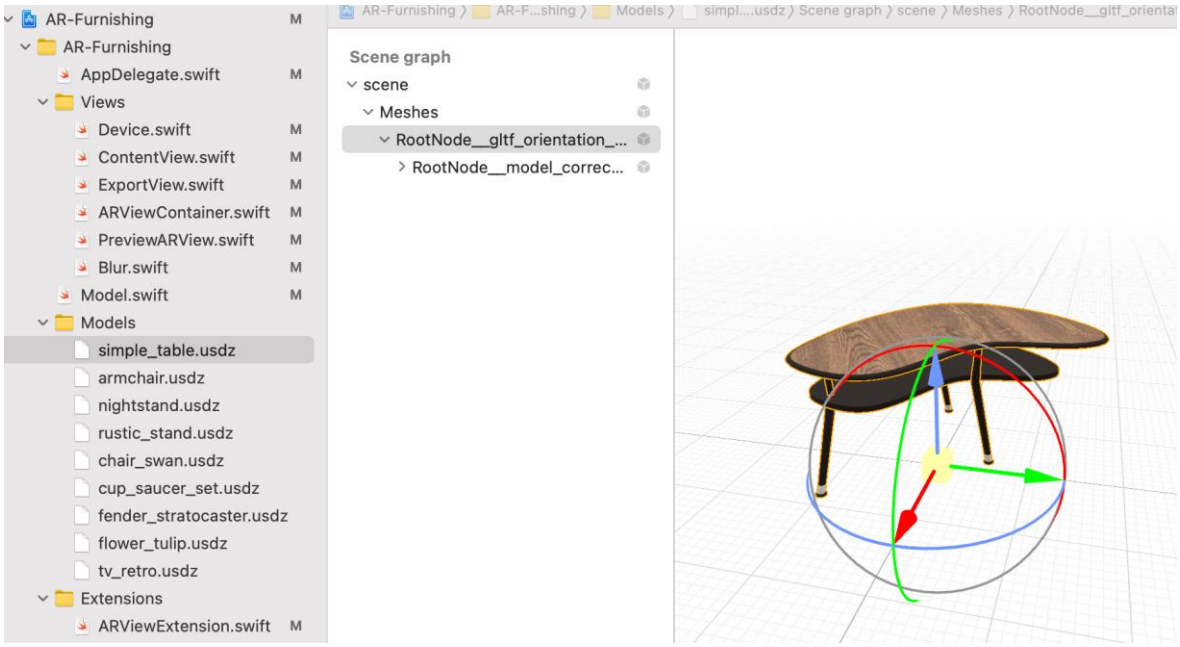
```
            switch loadCompletion {
            case .finished: break
            case .failure(let error):
               print("Unable to load model entity for \(name). Error: \(error.localizedDescription)")
            }

         }, receiveValue: { modelEntity in
            self.entity = modelEntity
            self.entity?.generateCollisionShapes(recursive: true)
         })
   }
}
```

Тобто ми використовуємо моделі з розширення .uszd яке є спеціальним розширенням, яке оголосило про створення Apple та Pixar і наразі використовується Apple як основний носій інформації про 3D об’єкти.

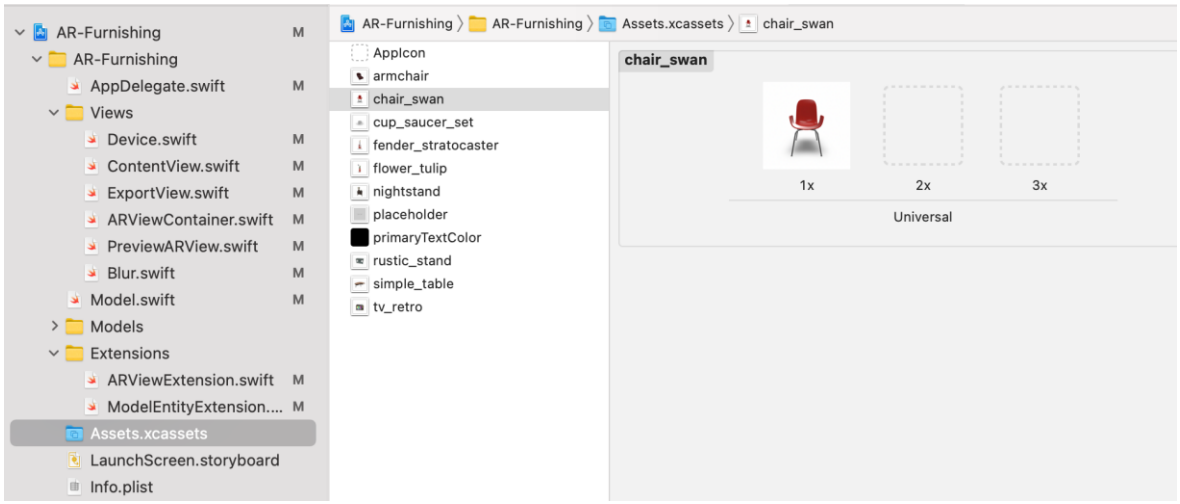




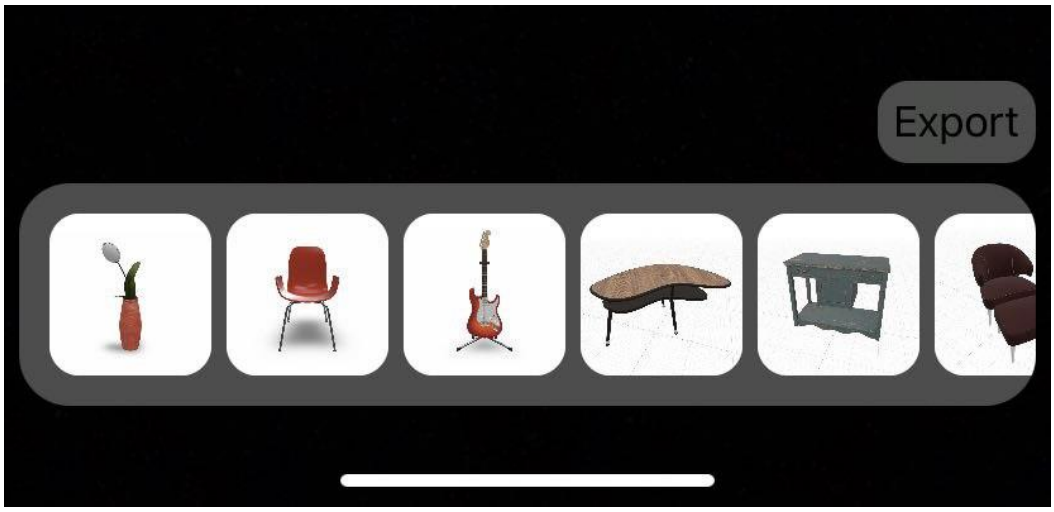
Є моделі, які дуже детально промальовані, але зазвичай вони є платними, але є й простіші моделі. Звичайно більш деталізовані модельки будуть виглядати краще на остаточній картинці.



Також ми не просто маємо добавити модельки, але й зробити щоб візуально було зрозуміло, що користувач хоче вибрати, тому в ассетах(активах) нам потрібно добавити вид модельки в 2D.



Отож, після всіх цих маніпуляцій поглянемо як саме буде виглядати даний інтерфейс в працюючій програмі.



Після цього нам потрібно зробити так, щоб ми могли вибирати де саме ми хочемо розміщувати наш об’єкт. Для цього використовуємо магію AR технологій, які вміють знаходити та визначати межі об’єктів, в нашому випадку горизонтальних поверхонь.

```
import Foundation
import RealityKit
import ARKit

class PreviewARView: ARView, ARSessionDelegate {

   var previewAnchor: AnchorEntity? = nil

   func session(_ session: ARSession, didUpdate frame: ARFrame) {
      if previewAnchor != nil {
         if let raycastResult = self.raycast(from: self.center, allowing: .estimatedPlane, alignment: .horizontal).first {
            previewAnchor?.anchoring = AnchoringComponent(.world(transform: raycastResult.worldTransform))
         }
      }
   }
}
```

Тобто за допомогою променя камера визначає межі вертикальних поверхонь, і створює якір до якого ми прив’яжемо модель, яку ми

вибрали. Тут же робимо можливість керувати даним об’єктом після встановлення, тобто обертати, переміщати, зменшувати або збільшувати.

```
import SwiftUI
import RealityKit
import ARKit

struct ARViewContainer: UIViewRepresentable {
   @Binding var selectedModel: Model?
   @Binding var modelConfirmedForPlacement: Model?

   func makeUIView(context: Context) -> PreviewARView {
      let arView = PreviewARView(frame: .zero, cameraMode: .ar, automaticallyConfigureSession: true)

      let config = ARWorldTrackingConfiguration()
      config.planeDetection = .horizontal
      config.environmentTexturing = .automatic

      let previewAnchor = AnchorEntity(plane: .horizontal)

      arView.scene.addAnchor(previewAnchor)
      arView.previewAnchor = previewAnchor

      arView.session.delegate = arView
      arView.session.run(config)

      arView.enableObjectRemoval()

      return arView
   }

   func updateUIView(_ uiView: PreviewARView, context: Context) {
      if let model = selectedModel, let modelEntity = model.entity?.clone(recursive: true) {
         modelEntity.fixScale(for: model.name)

         uiView.previewAnchor?.addChild(modelEntity)
```

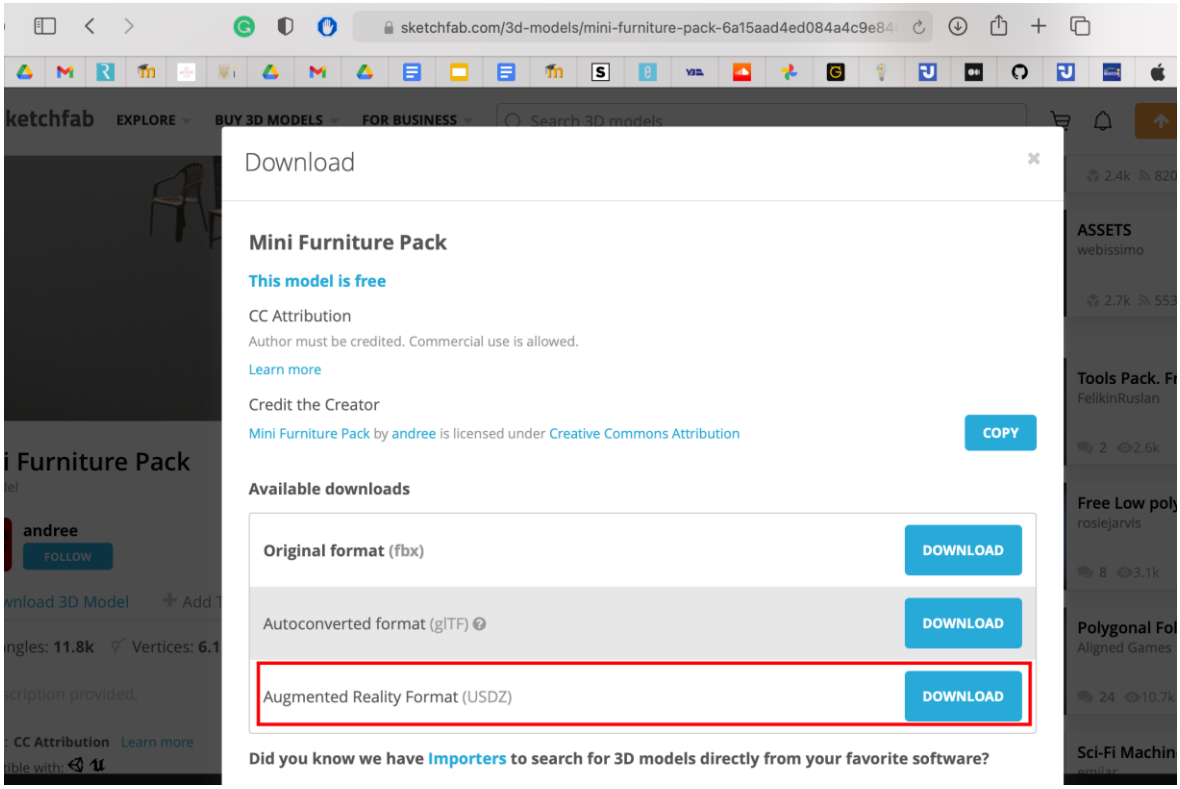
```
        } else {
            uiView.previewAnchor?.children.removeAll()
        }

        if let model = modelConfirmedForPlacement {
            if let modelEntity = model.entity?.clone(recursive: true) {
                DispatchQueue.main.async {
                    print("Adding model: \(model.name)")

                    let anchorEntity = AnchorEntity(plane: .horizontal)

                    modelEntity.fixScale(for: model.name)

                    anchorEntity.addChild(modelEntity)

                    uiView.installGestures([.translation, .rotation, .scale], for: modelEntity)

                    uiView.scene.addAnchor(anchorEntity)
                }
            }

            DispatchQueue.main.async {
                modelConfirmedForPlacement = nil
            }
        }
    }
}
```

Для того, щоб видалити об’єкт який вже розміщений пишемо додаток, який буде видаляти при довгому нажатті об’єкт, попередньо вказуючи скільки секунд має пройти щоб об’єкт видалився.

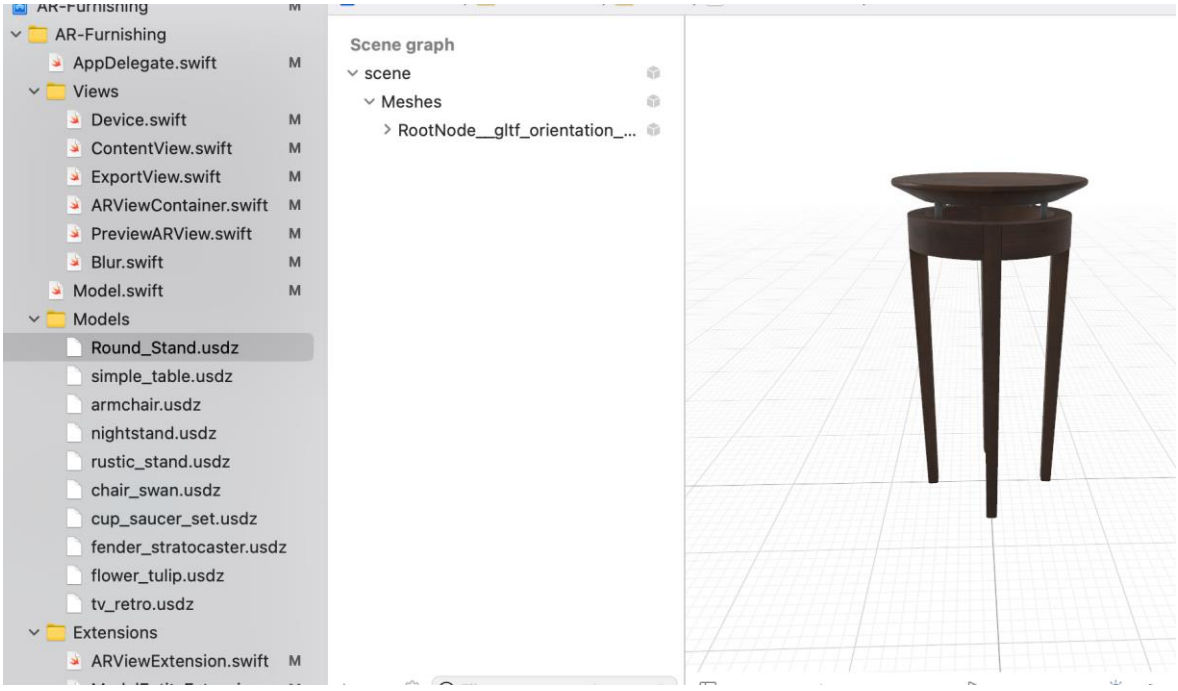
```
import Foundation
import RealityKit
import ARKit

extension ARView {
    func enableObjectRemoval() {
```

```
    let longPressGestureRecognizer = UILongPressGestureRecognizer(target:
self, action: #selector(handleLongPress(recognizer:)))
        longPressGestureRecognizer.minimumPressDuration = 0.5

        self.addGestureRecognizer(longPressGestureRecognizer)
    }

  @objc func handleLongPress(recognizer: UILongPressGestureRecognizer)
{
        let location = recognizer.location(in: self)

        if let entity = self.entity(at: location) {
            if let anchorEntity = entity.anchor {
                anchorEntity.removeFromParent()
            }
        }
    }
}
```

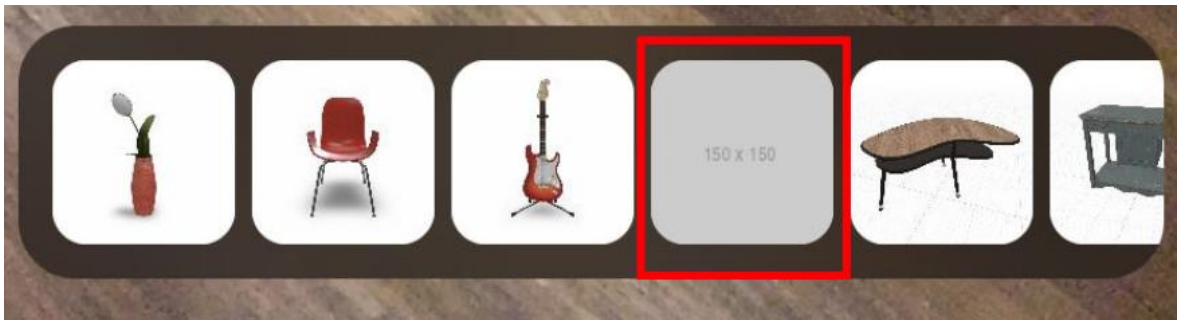
Якщо ми хочемо добавити якусь модель, то нам потрібно знайти її в розширенні .usdz та скачати



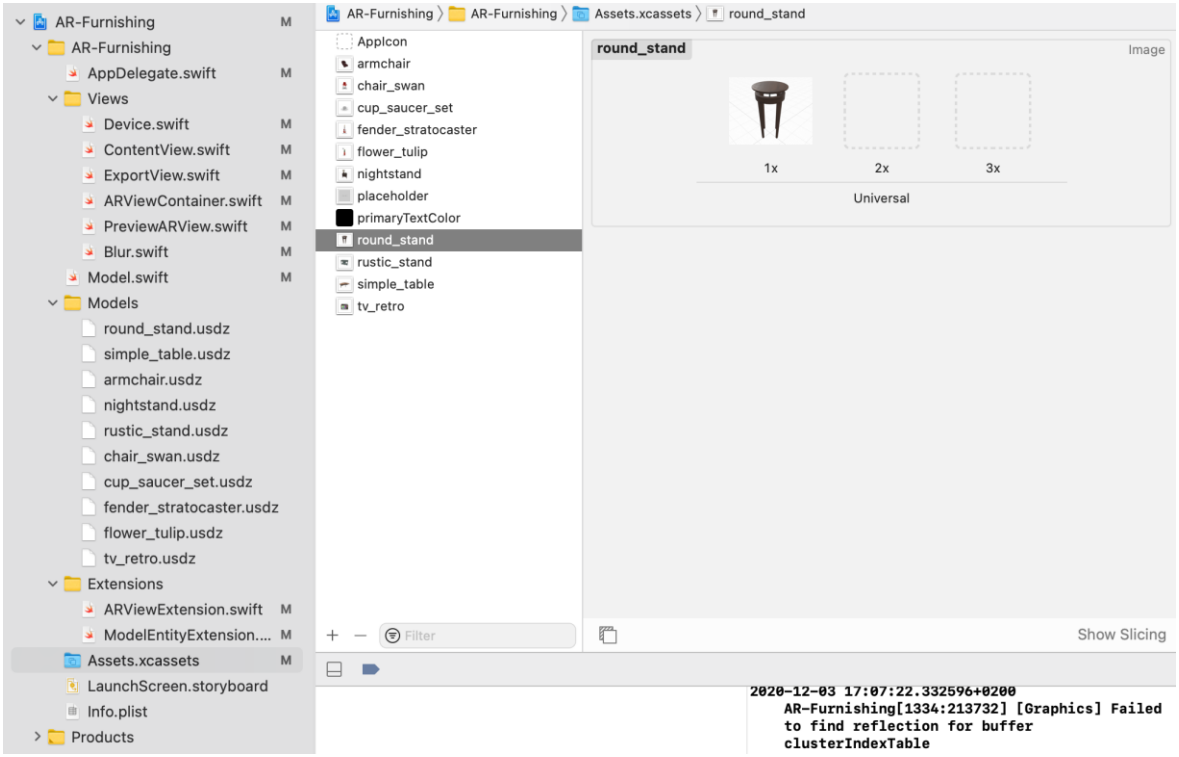
І потім цей файл додаємо до моделей



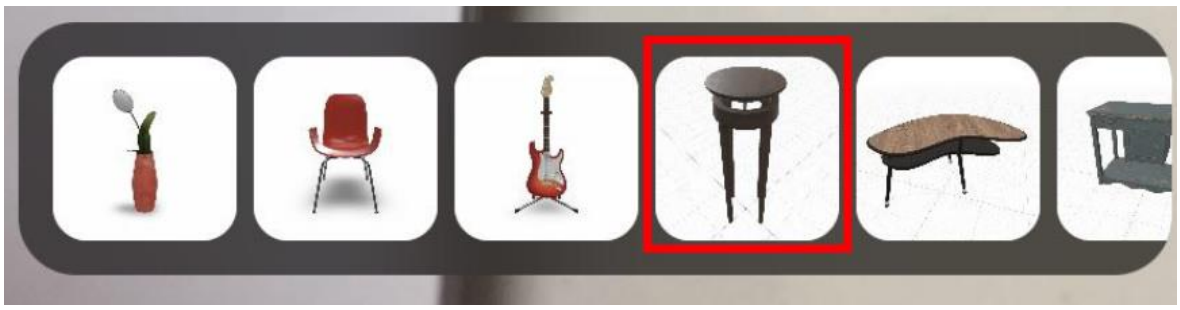
І, оскільки ми не вибрали фото для передогляду самої моделі, то наразі воно в нас показує спейсхолдер.



Для цього добавляємо фото в ассету(активу).



І тепер воно буде відображатись:



ВИСНОВКИ

Магістерська робота присвячена питанням візуалізації та обробки інформації. Зокрема вдосконаленню візуалізації тривимірних об'єктів засобами технології доповненої реальності. Концепція доповненої реальності пропонує більш досконалий користувальницький інтерфейс для візуалізації за рахунок сукупності природних способів управління зміною ракурсу об'єкта та візуалізації в реальному контексті.

При відтворенні системою комп'ютерної графіки умов монокулярного спостереження, об'ємність зображення, просторове положення об'єктів сприймаються на синтезованому зображенні завдяки лінійної перспективі, загораживание одних об'єктів іншими, характером тіней і зміни тону по полю зображення. Істотне значення для сприйняття обсягу і простору має попередній досвід спостереження, завдяки якому користувач мимоволі "добудовує" об'ємну структуру спостерігається сцени. Таким чином, пропонована доповненою реальністю візуалізація в реальному знайомому для користувача оточенні сприяє кращому сприйняттю тривимірних об'єктів.

Особливістю роботи є орієнтація на масове впровадження доповненої реальності. Для цього в якості платформ для застосування результатів дослідження розглядалися сучасні масово доступні мобільні пристрої: смартфони і планшети компанії Apple.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ronald T. Azuma Survey of Augmented Reality, pp. 355—385, August 1997
2. Johnson, Joel. "The Master Key": L. Frank Baum envisions augmented reality glasses in 1901 *Mote & Beam* 10 September 2012.
3.
4. Louis B. Rosenberg. "The Use of Virtual Fixtures As Perceptual Overlays to Enhance Operator Performance in Remote Environments." Technical Report AL-TR-0089, USAF Armstrong Laboratory (AFRL)
5. Outdoor AR // https://www.youtube.com/watch?v=jL3C-OVQKWU
6. Tim Cook Interview to “Independent” // https://www.independent.co.uk/life-style/gadgets-and-tech/features/apple-iphone-tim-cook-interview-features-new-augmented-reality-ar-arkit-a7993566.html
7. Consumer Mixed Reality: Emerging Opportunities, Vendor Strategies & Market Forecasts 2019-2024 // https://www.juniperresearch.com/researchstore/devices-technology/consumer-mixed-reality-market-research-report
8. Assisting on surgery // https://www.youtube.com/watch?v=aTOoBwfqBe0
9. Classes with using of AR technology // https://www.youtube.com/watch?v=-DYqlaMWTVg
10. How Augmented Reality Will Change Education Completely // https://www.youtube.com/watch?v=5AjxGqzqQ54
11. Civil aviation // https://vimeo.com/210800728
12. Rotterdam’s most extraordinary trees // https://www.themayor.eu/en/discover-rotterdams-unique-trees-with-this-augmented-reality-trail

13. http://wearcam.org/PhenomenalAugmentedReality.pdf // http://wearcam.org/PhenomenalAugmentedReality.pdf
14. AR Furnishing // https://www.youtube.com/watch?v=DSYeN624ick
15. Online Shopping // https://www.youtube.com/watch?v=toJFwFC5AeY
16. Pokemon GO hysteria // https://www.youtube.com/watch?v=DNjpqm73GBY
17. Best Augmented Reality Marketing Experiences // https://www.youtube.com/watch?v=3kZcTvNsT0Y
18. https://developer.apple.com
19. https://sketchfab.com/feed

AR Furnishing

AppDelegate

```
import UIKit
import SwiftUI

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        // Create the SwiftUI view that provides the window contents.
        let contentView = ContentView()

        // Use a UIHostingController as window root view controller.
        let window = UIWindow(frame: UIScreen.main.bounds)
        window.rootViewController = UIHostingController(rootView:
contentView)
        self.window = window
        window.makeKeyAndVisible()

        return true
    }

}
```

ARViewContainer

```
import SwiftUI
import RealityKit
import ARKit
```

```
struct ARViewContainer: UIViewRepresentable {
    @Binding var selectedModel: Model?
    @Binding var modelConfirmedForPlacement: Model?

    func makeUIView(context: Context) -> PreviewARView {
        let arView = PreviewARView(frame: .zero, cameraMode: .ar,
automaticallyConfigureSession: true)

        let config = ARWorldTrackingConfiguration()
        config.planeDetection = .horizontal
        config.environmentTexturing = .automatic

        let previewAnchor = AnchorEntity(plane: .horizontal)

        arView.scene.addAnchor(previewAnchor)
        arView.previewAnchor = previewAnchor

        arView.session.delegate = arView
        arView.session.run(config)

        arView.enableObjectRemoval()

        return arView
    }

    func updateUIView(_ uiView: PreviewARView, context: Context) {
        if let model = selectedModel, let modelEntity =
model.entity?.clone(recursive: true) {
            modelEntity.fixScale(for: model.name)

            uiView.previewAnchor?.addChild(modelEntity)
        } else {
            uiView.previewAnchor?.children.removeAll()
        }

        if let model = modelConfirmedForPlacement {
```

```
            if let modelEntity = model.entity?.clone(recursive: true) {
                DispatchQueue.main.async {
                    print("Adding model: \(model.name)")

                    let anchorEntity = AnchorEntity(plane: .horizontal)

                    modelEntity.fixScale(for: model.name)

                    anchorEntity.addChild(modelEntity)

                    uiView.installGestures([.translation, .rotation, .scale], for:
modelEntity)

                    uiView.scene.addAnchor(anchorEntity)
                }
            }

            DispatchQueue.main.async {
                modelConfirmedForPlacement = nil
            }
        }
    }
}
```

Blur

```
import SwiftUI

struct Blur: UIViewRepresentable {

    var style: UIBlurEffect.Style = .systemMaterial

    func makeUIView(context: Context) -> UIVisualEffectView {
        return UIVisualEffectView(effect: UIBlurEffect(style: style))
    }

    func updateUIView(_ uiView: UIVisualEffectView, context: Context) {
```

```
        uiView.effect = UIBlurEffect(style: style)
    }
}
```

ContentView

```
import SwiftUI

struct ContentView: View {

    @State private var modelPlacementEnabled: Bool = false
    @State private var selectedModel: Model?
    @State private var modelConfirmedForPlacement: Model?

    private var models: [Model] = {
        let filemanager = FileManager.default

        guard let path = Bundle.main.resourcePath, let files = try?
filemanager.contentsOfDirectory(atPath: path)
        else { return [] }

        var availableModels: [Model] = []
        for filename in files where filename.hasSuffix("usdz") {
            let modelName = filename.replacingOccurrences(of: ".usdz", with: "")
            let model = Model(name: modelName)

            availableModels.append(model)
        }

        return availableModels
    }()

    var body: some View {
        NavigationView {
            ZStack(alignment: .bottom) {
```

```
            ARViewContainer(selectedModel: $selectedModel,
modelConfirmedForPlacement:
$modelConfirmedForPlacement).edgesIgnoringSafeArea(.all)

            if modelPlacementEnabled {
                PlacementButtonsView(modelPlacementEnabled:
$modelPlacementEnabled, selectedModel: $selectedModel,
modelConfirmedForPlacement: $modelConfirmedForPlacement)
            } else {
                ModelPickerView(modelPlacementEnabled:
$modelPlacementEnabled, selectedModel: $selectedModel, models: models)
            }
        }
    }
}

struct ModelPickerView: View {
    @Binding var modelPlacementEnabled: Bool
    @Binding var selectedModel: Model?

    @ObservedObject var device = Device()

    var models: [Model]

    var body: some View {
        VStack {
            HStack {
                Spacer()
                NavigationLink(destination: ExportView()) {
                    Text("Export")
                        .font(.body)
                        .foregroundColor(Color("primaryTextColor"))
                        .padding(6)
                }
                .background(Blur(style: .regular))
                .cornerRadius(12)
```

```
            }

            ScrollView(.horizontal, showsIndicators: false) {
                HStack(spacing: 6) {
                    ForEach(0 ..< models.count) { index in
                        Button(action: {
                            selectedModel = models[index]

                            modelPlacementEnabled = true
                        }) {
                            Image(uiImage: models[index].thumbnail)
                                .resizable()
                                .frame(height: 64)
                                .aspectRatio(1/1, contentMode: .fit)
                                .background(Color.white)
                                .cornerRadius(12)
                                .rotationEffect(device.getImageOrientationAngle())
                                .animation(Animation.easeInOut)
                        }
                        .buttonStyle(PlainButtonStyle())
                    }
                }
                .padding(12)
            }
            .background(Blur(style: .regular))
            .cornerRadius(20)
        }
        .padding(6)
    }
}

struct PlacementButtonsView: View {
    @Binding var modelPlacementEnabled: Bool
    @Binding var selectedModel: Model?
    @Binding var modelConfirmedForPlacement: Model?

    var body: some View {
```

```
            HStack {
                Button(action: {
                    resetPlacementState()
                }) {
                    Image(systemName: "xmark")
                        .frame(width: 64, height: 64)
                        .font(.title)
                        .foregroundColor(Color.red)
                        .background(Color.white.opacity(0.5))
                        .cornerRadius(32)
                        .padding(12)
                }

                Button(action: {
                    modelConfirmedForPlacement = selectedModel

                    resetPlacementState()
                }) {
                    Image(systemName: "checkmark")
                        .frame(width: 64, height: 64)
                        .font(.title)
                        .foregroundColor(Color.white)
                        .background(Color.white.opacity(0.5))
                        .cornerRadius(32)
                        .padding(12)
                }
            }
        }

        func resetPlacementState() {
            selectedModel = nil
            modelPlacementEnabled = false
        }
    }

    struct ContentView_Previews: PreviewProvider {
        static var previews: some View {
```

```
        ContentView()
    }
}
```

DeviceOrientation

```
import Foundation
import UIKit
import SwiftUI

class Device: ObservableObject {

    @Published var orientation: UIDeviceOrientation =
UIDevice.current.orientation

    private var _observer: NSObjectProtocol?

    init() {
        _observer = NotificationCenter.default.addObserver(forName:
UIDevice.orientationDidChangeNotification, object: nil, queue: nil) { _ in
            self.orientation = UIDevice.current.orientation
        }
    }

    deinit {
        if let observer = _observer {
            NotificationCenter.default.removeObserver(observer)
        }
    }

    func getImageOrientationAngle() -> Angle {
        switch orientation {
        case .landscapeLeft:
            return Angle(degrees: 90)
        case .landscapeRight:
            return Angle(degrees: -90)
        default:
```

```
            return Angle(degrees: 0)
        }
    }
}
```

ExportView

```
import SwiftUI

struct ExportView: View {

    var body: some View {
        ZStack(alignment: .bottom) {
        }
    }
}

struct ExportView_Previews: PreviewProvider {
    static var previews: some View {
        ExportView()
    }
}
```

Models

```
import UIKit
import RealityKit
import Combine

class Model {
    var name: String
    var thumbnail: UIImage
    var entity: ModelEntity?

    private var cancellable: AnyCancellable? = nil

    init(name: String) {
```

```
        self.name = name
        self.thumbnail = UIImage(named: name) ?? UIImage(named:
"placeholder")!

        let filename = name + ".usdz"
        self.cancellable = ModelEntity.loadModelAsync(named: filename)
            .sink(receiveCompletion: { loadCompletion in
                switch loadCompletion {
                case .finished: break
                case .failure(let error):
                    print("Unable to load model entity for \(name). Error:
\(error.localizedDescription)")
                }

            }, receiveValue: { modelEntity in
                self.entity = modelEntity
                self.entity?.generateCollisionShapes(recursive: true)
            })
    }
}
```

PreviewARView

```
import Foundation
import RealityKit
import ARKit

class PreviewARView: ARView, ARSessionDelegate {

    var previewAnchor: AnchorEntity? = nil

    func session(_ session: ARSession, didUpdate frame: ARFrame) {
        if previewAnchor != nil {
            if let raycastResult = self.raycast(from: self.center, allowing:
.estimatedPlane, alignment: .horizontal).first {
```

```
                previewAnchor?.anchoring =
AnchoringComponent(.world(transform: raycastResult.worldTransform))
            }
        }
    }
}
```