МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

С. О. Семеріков

М. М. Мінтій

ПРОЄКТУВАННЯ ЦИФРОВИХ ОСВІТНІХ РЕСУРСІВ ІЗ ДОПОВНЕНОЮ РЕАЛЬНІСТЮ

Навчальний посібник із курсу
“Інноваційні цифрові технології в освіті”

Кривий Ріг
2023

УДК 378::[372.8::[501+62+004]]+004.946

Проєктування цифрових освітніх ресурсів із доповненою реальністю : навчальний посібник із курсу “Інноваційні цифрові технології в освіті” / С. О. Семеріков, М. М. Мінтій. – Кривий Ріг, 2023.

Зміст

Вступ

Програмувати доповнену реальність (augmented reality – AR) – інноваційно (модно, цікаво, корисно та ін.) останні 70 років, і вибір JavaScript в якості мови програмування – виключно данина часу. Але вибір мови визначає й вибір засобів розробки.

Розробка веб-додатків доповненої реальності (WebAR) відрізняється від інших способів розробки тим, що є крос-платформовою і не вимагає встановлення розроблених додатків, адже це просто звичайні веб-сторінки.

Наразі найвідоміша у світі бібліотека для розробки WebAR – AR.js (`https://ar-js-org.github.io/AR.js-Docs/`), проте XiyKim Юен (HiuKim Yuen, `https://www.youtube.com/channel/UC-JyA1Z1-p0wgxj5WEX56wg/featured`), один із її розробників, створив нову бібліотеку під назвою MindAR (`https://hiukim.github.io/mind-ar-js-doc/`) – більш компактну та технологічно розвинену. Це основні бібліотеки, які ми будемо використовувати для відстеження об'єктів та створення ефектів. Отримані знання, безумовно, можуть бути перенесені й на інші, зокрема, комерційні засоби.

WebXR – це JavaScript API для створення імерсивних ресурсів у браузерах: AR (augmented reality – доповненої реальності) та VR (virtual reality – віртуальної реальності). Будучи стандартом, він, ймовірно, стане однією з найбільш досконалих веб-технологій 2020-х років. Це, безумовно, те, що ви повинні взяти на озброєння, якщо ви серйозно ставитеся до WebAR або WebVR.

Ви не зможете розробити жодних серйозних AR додатків без опанування фреймворків для 3D-візуалізації, таких як Three.js (`https://threejs.org/`) та A-Frame (`https://aframe.io`).

A-Frame та AR.js – ці API, фактично, унікальні засоби швидкого прототипування, і значна частина програми з їх використанням – це HTML-подібний код, який використовує JavaScript на сервері. A-Frame використовується для створення сцен, об'єктів, анімації та інших 3D-елементів у веб-браузері. AR.js надає можливість відслідковувати маркер і надає можливість сцені, сконструйованій за допомогою A-Frame, відображатися прямо на маркері. Three.js та ARToolKit – своєрідний кістяк, який використовуються багато інших бібліотек мовою JavaScript. Three.js викори-

стовує WebGLRenderer, що надає можливість створення якісних 3D-сцени безпосередньо у браузері. На відміну від A-Frame, Three.js вимагає явного використання JavaScript на боці клієнта.

Розробка програмного забезпечення із доповненою реальністю часто вимагає зовнішнього пристрою для тестування. Полегшити цей процес можна за допомогою віддаленого налагодження та заздалегідь записаних відео, що імітують веб-камеру.

Важливою родзинкою цього курсу є те, що він допоможе зміцнити знання про те, як працює AR в середовищі браузера.

У цьому курсі ми також дізнаємося, як за допомогою TensorFlow.js (`https://www.tensorflow.org/js`) інтегрувати моделі машинного навчання у WebAR додатки для створення високоінтерактивних і цікавих ефектів, наприклад, використання жестів рук або міміки для управління AR-контентом.

Посібник структурований у такий спосіб.

Розділ 1 допоможе налаштувати середовище розробки мовою JavaScript – локальний веб-сервер, віддалені доступ та налагодження, а також розгортання програм на зовнішніх серверах.

Розділ 2 уведе до фундаментальних концепцій віртуальної та доповненої реальності, основ 3D-рендерингу з використанням Three.js, управління вебкамерою та відстеження об'єктів.

Розділ 3 занурить у проєктування 3D-сцени для віртуальної та доповненої реальності за допомогою A-Frame.

1 Засоби розробки

1.1 Вступ до JavaScript

Для початку застосування JavaScript достатньо знати мінімальний синтаксис цієї мови:

1. *Змінні* у JavaScript створюються з літер, цифр, знаків долару та підкреслення:

а) при першому зверненні до них:

```
x = 1 // створити змінну x та надати їй значення 1
```

б) за допомогою ключового слова `var`
так:

```
var x // створити змінну x
x = 1 // надати значення змінній
```

або так

```
var x = 1 // створити змінну x та надати їй значення 1
```

Змінні, створені без використання `var`, стають глобальними. Змінні також можуть бути оголошені за допомогою `let` (для змінної рівня блоку) та `const` (для сталої).

2. *Коментарі* створюються аналогічно до C++:

```
// однорядковий коментар
/*
багаторядковий
коментар
*/
```

3. *Прості типи даних*:

- рядковий (`string`) – визначається подвійними або одинарними лапками і використовується для символьних даних;

- числовий (`number`) – визначається відсутністю лапок і використовується для дійсних чисел (наприклад, `345` – ціле десяткове, `34.5` – число з плаваючою точкою, `0b1011` – ціле двійкове, `0o377` – вісімкове, `0xFF` – шістнадцяткове, `Infinity` – $+\infty$, `NaN` – помилкове число);
- логічний (`boolean`) – визначається відсутністю лапок і використовується для значень `true = 1` або `false = 0`;
- символьний (`Symbol`) – тип унікального незмінного ідентифікатору;
- невизначений (`undefined`) – тип будь-якої неініціалізованої змінної (такої, якій не було надане значення).

4. *Спеціальні типи даних*:

- порожній (`null`) – відсутність даних у оголошеній змінній;
- об'єкт (`object`) – програмний об'єкт (посилання на нього);
- функція (`function`) – визначення функції.

5. *Основні оператори*:

+	додавання як бінарний, перетворення рядка на число як унарний;
-	віднімання як бінарний, зміна знаку як унарний;
*	множення;
/	ділення;
%	ділення за модулем (остача);
++	інкремент (збільшення на 1);
=	надання значення;
+=	додати та надати значення;
-=	відняти та надати значення;
*=	помножити та надати значення;
/=	поділити та надати значення;
%=	знайти остачу від ділення та надати значення;
+=	додати та надати значення;
==	дорівнює;
!=	не дорівнює;

>	більше;
>=	більше або рівний;
<	менше;
<=	менше або рівний;
===	ідентичний (дорівнює та одного типу);
!==	не ідентичний;
!	логічне заперечення;
||	логічна диз'юнкція;
&&	логічна кон'юнкція;
&	побітова кон'юнкція;
|	побітова диз'юнкція;
^	бінарна виключна диз'юнкція;
<<	побітовий зсув вліво;
>>	побітовий зсув вправо (із збереженням знаку);
>>>	побітовий зсув вправо (без збереження знаку);
~	побітове заперечення.

6. *Визначення функції*:

```
function ім'я_функції(параметр1, параметр2, ..., параметрn)
{
  оператори
  return значення_що_повертається;
}
```

або

```
var ім'я_функції = function(парам1, парам2, ..., парамn)
{
  оператори
  return значення_що_повертається;
}
```

або

```
var ім'я_функції = new Function('парам1', 'парам2', ..., 'парамn',
  'оператори; return значення_що_повертається');
```

7. *Умовний вираз*:

```
if(умова)
{
  оператори1
}
else  // інакше, якщо умова не виконалась
{
  оператори2
}
```

або

```
результат = умова ? оператори1 : оператори2;
```

8. *Оператор вибору* надає можливість порівняти одну змінну з великою кількістю констант. Наприклад:

```
var  a ;
switch ( a )
{
  case  1 :  // якщо a = 1
    оператори
    [ break ;  ]
  case  2 :  // якщо a = 2
    оператори
    [ break ;  ]
  default :  // якщо a = 3
    оператори
    [ break ;  ]
}
```

`case` порівнює змінну, зазначену в `switch` (змінна). `break` перериває виконання `case` або `default`, тобто якщо він буде відсутнім при виконанні хоча б першого `case`, виконаються всі наступні та `default`. `default` виконається, тільки якщо не виконається жоден із операторів `case`.

9. *Цикли*:

- `while` – цикл з передумовою, який триватиме до того моменту, коли умова не перестане виконуватись:

```
while(умова)
{
  оператори
}
```

- `do ... while` – цикл з післяумовою, який відрізняється від циклу while тим, що умова перевіряється наприкінці виконання блоку:

```
do {
  оператори
}while(умова)
```

- `for` – ітераційний цикл:

```
for(var змінна = початкове_значення; умова; крок циклу) {
  оператори
}
```

або

```
for (var ім'я_властивості in деякий_об'єкт) {
  // дії за допомогою деякий_об'єкт[ім'я_властивості];
}
```

10. Типи JavaScript поділяються на примітивні та об'єктні. Об'єкти можуть розглядатися як асоціативні масиви або хеші, тому часто реалізуються з використанням цих структур даних. *Стандартні об'єкти*: `Array`, `Date`, `Error`, `Math`, `Boolean`, `Function`, `Number`, `Object`, `RegExp`, `String`. Інші об'єкти – це "хост-об'єкти", що визначаються не мовою, а середовищем виконання (наприклад, у браузері типові хост-об'єкти належать до DOM).

Об'єкти можуть бути створені за допомогою конструктора або літерала об'єкта (останнє є основою об'єктної нотації JavaScript – JSON):

```
var anObject = new Object(); // конструктор
// літерали
var objectA = {};
var objectB = {index1:'значення 1', index2:'значення 2'};
```

Як було показано далі, для доступу до даних та методів об'єкту застосовується оператор «точка» (.).

Все, що стосується синтаксису JavaScript/ECMAScript, можна знайти у багатьох джерелах – наприклад, якісних відеолекціях Дугласа Крокфорда (`https://youtu.be/playlist?list=PLEzQf147-uEpvTa1bHDNlxUL2klHUMHJu`).

1.2 Налаштування веб-серверу Simple Web Server

Для розробки мовами HTML та JavaScript основними засобами розробки є простий текстовий редактор і веб-браузер (рис. 1.1), в якому можна відкрити звичайну веб-сторінку HTML, збережену локально.

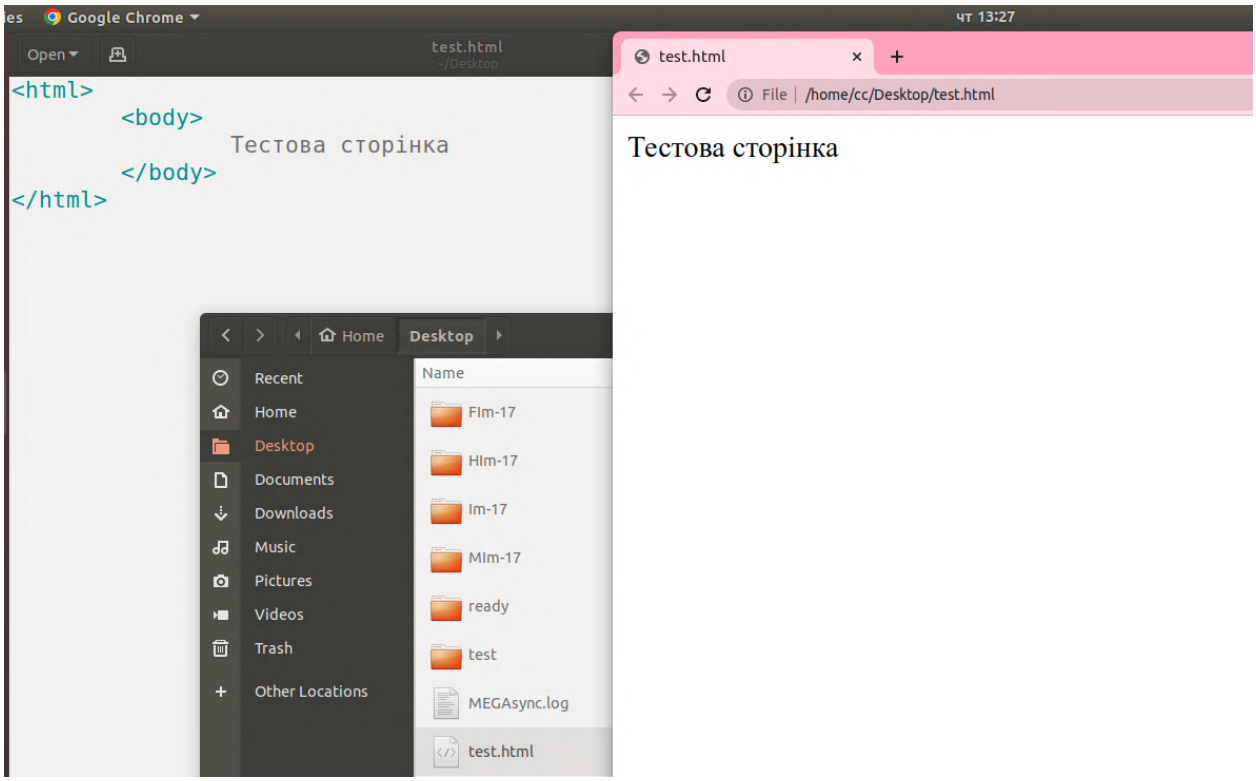

Рис. 1.1. Текстовий редактор – основний засіб розробки.

Однак це може не спрацювати для додатків, які потребують використання камери. Крім того, вам може знадобитися час від часу тестувати додатки на власних мобільних пристроях, тому краще встановити локальний веб-сервер.Один з простих, але потужних веб-серверів – Simple Web Server (`https://simplewebserver.org/download.html`). Після встановлення необхідно запустити сервер, а потім обрати кореневий каталог для

веб-сторінок та обрати можливість доступу у локальній мережі, щоб інші пристрої також могли отримати доступ до веб-сторінки (рис. 1.2, 1.3); по завершенні роботи веб-сервер можна зупинити.

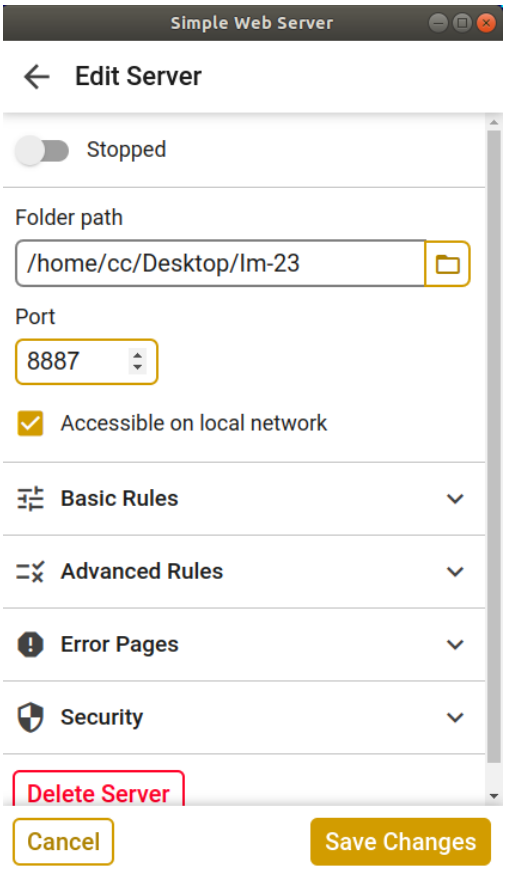

Рис. 1.2. Налаштування Simple Web Server.

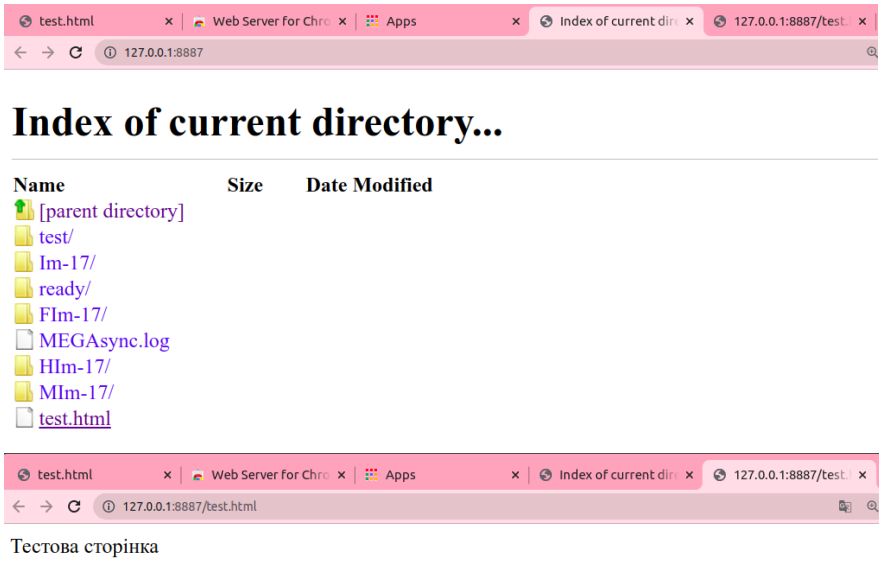

Рис. 1.3. Робота веб-серверу.

Корисним може бути вибір протоколу HTTPS у додаткових налашту-

ваннях – без його використання мобільний пристрій може не надати доступ до камери.

Технічно можна виконувати всю роботу з розробки та тестування безпосередньо на настільному браузері, але іноді все ж таки доцільно спробувати на мобільному телефоні (рис. 1.4).

Рис. 1.4. Доступ до веб-сторінки у локальній мережі з різних пристроїв.

1.3 Віддалений доступу до веб-сервера через ngrok

Якщо пристрої підключені до однієї локальної мережі, у якій немає брандмауера, проблем із доступом до веб-серверу немає. Однак, якщо точка доступу до мережі знаходиться за брандмауером, можна використовувати ngrok (`https://ngrok.com/`) для того, щоб виконати перенаправлення трафіку з порту, доступ до якого обмежений.

Після встановлення ngrok та створення облікового запису на сайті `https://ngrok.com/` необхідно зареєструвати агент ngrok (`https://dashboard.ngrok.com/get-started/your-authtoken`) та запустити його, вказавши в якості параметру протокол (наприклад, http) та номер порту, доступ до якого закриває брандмауера (наприклад, 8887) (рис. 1.5).

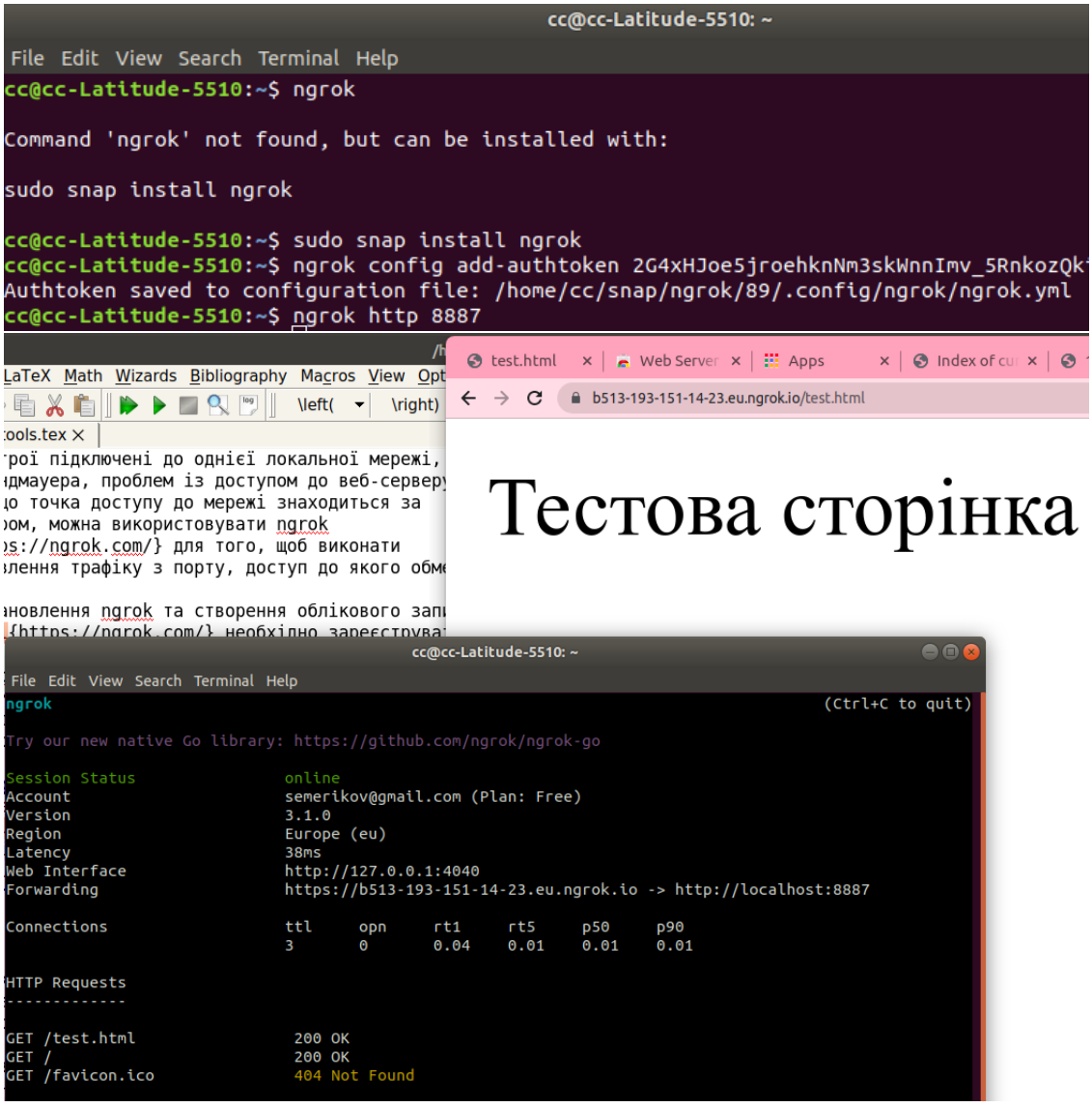

Рис. 1.5. Застосування ngrok для обходу брандмауера.

Після запуску ngrok надає посилання, яке глобально Інтернет доступне за протоколом HTTPS – але лише у той час, коли працюють одночасно локальний веб-сервер та перенаправлення ngrok (рис. 1.6).

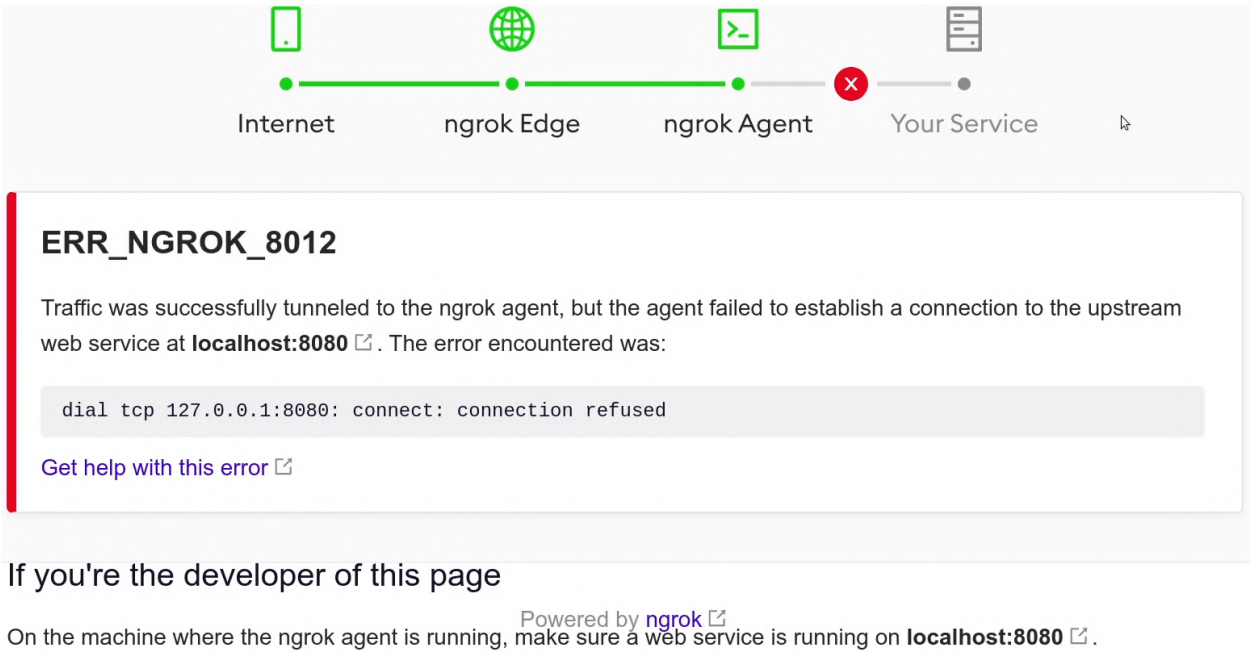

Рис. 1.6. Помилка: ngrok працює, локальний веб-сервер призупинений.

1.4 Віддалене налагодження через RemoteJS

Традиційно, налагодження веб-додатків передбачає перегляд консолі веб-браузера, куди виводяться повідомлення, що стосуються налагодження програми (рис. 1.7):

```
<html>
    <body>
        Тестова сторінка
        <script>
            console.log("Повідомлення");
        </script>
    </body>
</html>
```

Однак на мобільному пристрої це може бути не так просто. Тут допоможе RemoteJS (`https://remotejs.com/`) – натиснувши після переходу на сайт кнопку “Start Debugging”, отримаємо код агенту RemoteJS виду

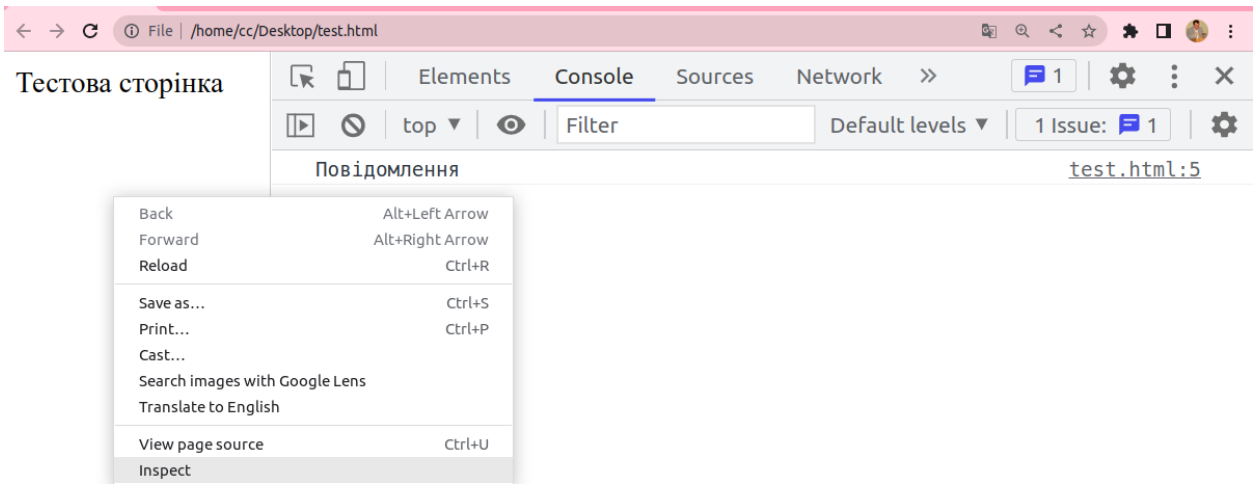

Рис. 1.7. Inspect – перегляд консолі веб-браузера.

```
<script
  data-consolejs-channel="9817ec3e-a3f7-fbe3-3836-e2e2d07d5c99"
  src="https://remotejs.com/agent/agent.js"></script>
```

Цей код необхідно скопіювати і вставити безпосередньо у веб-сторінку:

```
<html>
    <head>
    <script
    data-consolejs-channel="9817ec3e-a3f7-fbe3-3836-e2e2d07d5c99"
    src="https://remotejs.com/agent/agent.js">
    </script>
    </head>
    <body>
        Тестова сторінка
        <script>
            console.log("Повідомлення");
        </script>
    </body>
</html>
```

Після цього всі налагоджувальні повідомлення будуть надіслані на веб-сторінку з адресою `https://remotejs.com/viewer/agent_code`, де agent_code – значення змінної data-consolejs-channel (рис. 1.8).

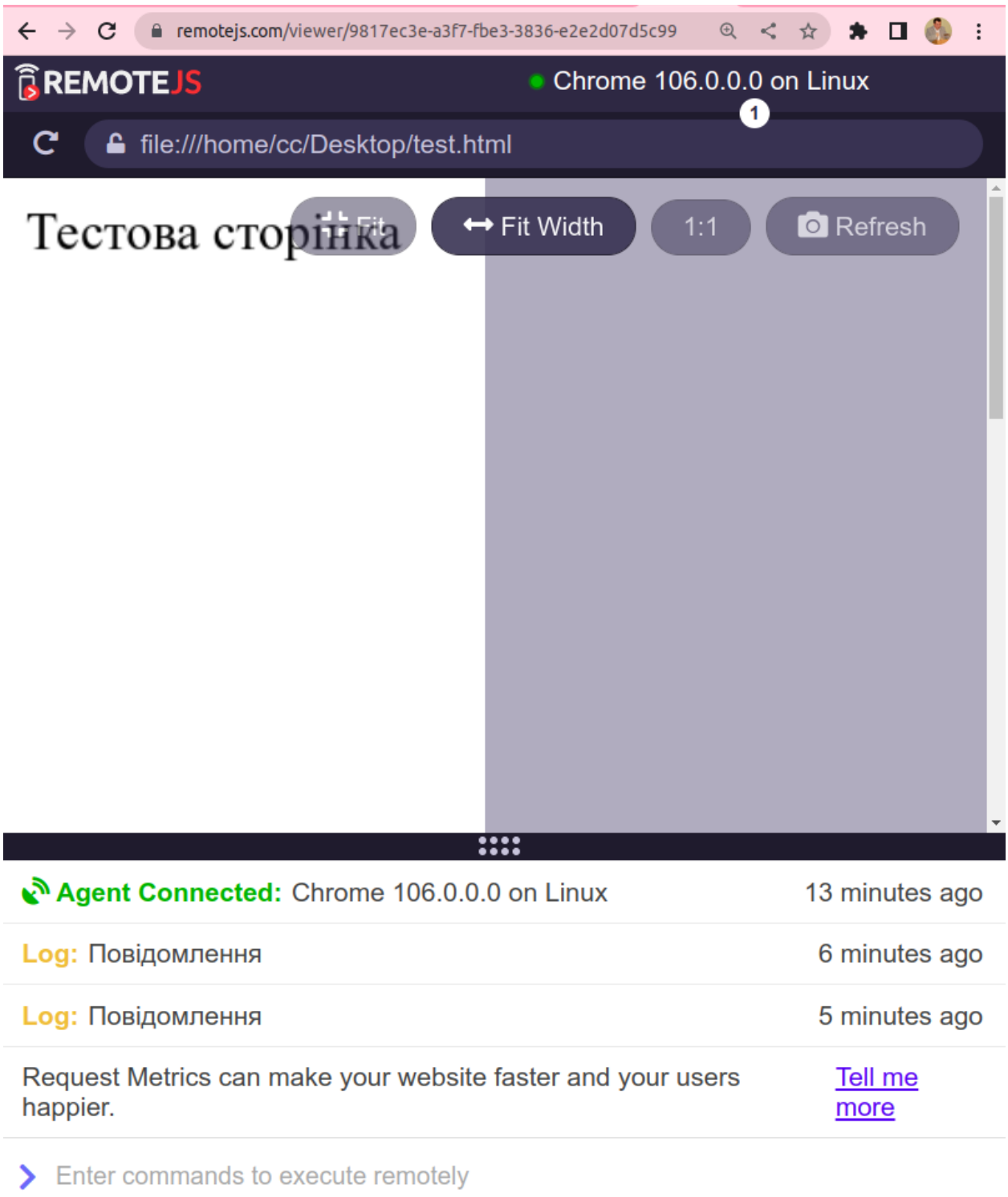

Рис. 1.8. Віддалене налагодження за допомогою агенту RemoteJS.

1.5 Розгортання програм на GitHub Pages

Для забезпечення постійного глобального доступу до розробленого програмне забезпечення файли HTML, JavaScript, зображень, бібліотеки тощо доцільно розмістити на зовнішньому хостингу, такому як GitHub Pages. Для цього необхідно:

1. Зареєструватись на GitHub:

   (а) перейти на сайт GitHub за адресою `https://github.com/`;

   (б) натиснути кнопку “Sign up”;

   (в) увести адресу електронної пошти, ім’я користувача та пароль;

   (г) натиснути кнопку “Create account”;

   (д) підтвердити електронну адресу, перейшовши за посиланням у листі, надісланому з GitHub.

2. Створити репозиторій:

   (а) увійти до облікового запису GitHub;

   (б) натиснути кнопку “Create repository”;

   (в) увести назву репозиторію;

   (г) обрати тип репозиторію: “Public” (публічний) або “Private” (приватний);

   (д) натиснути кнопку “Create a new repository” (рис. 1.9).

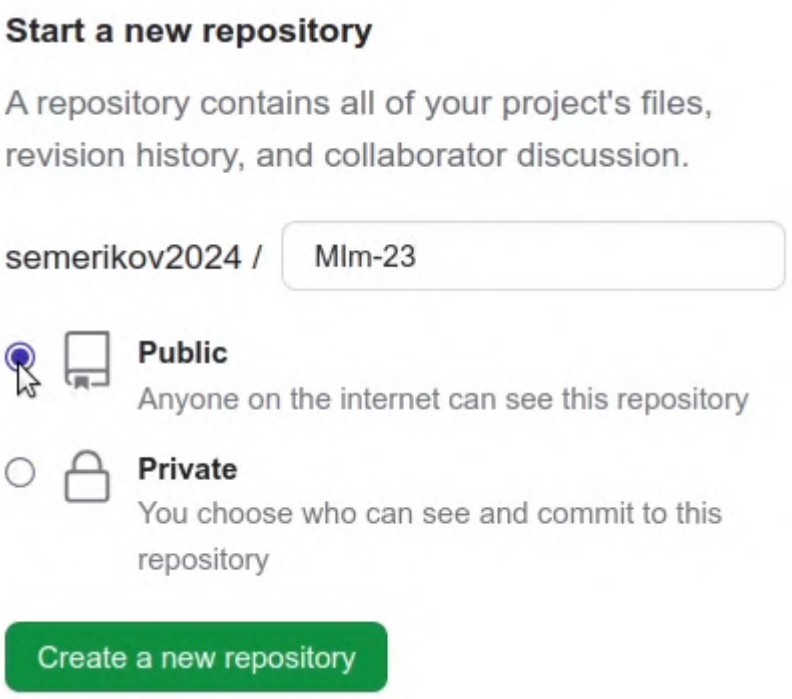

Рис. 1.9. Створення репозиторію GitHub.

3. Опублікувати вмісту репозиторію на GitHub Pages:

(а) відкрити репозиторій, вміст якого необхідно опублікувати;

(б) у розділі “Settings” натиснути посилання “Pages”;

(в) обрати тип розгортання “Deploy from a branch” (розгорнути з гілки репозиторію) та гілку (зазвичай це “main”) (рис. 1.10);

(г) натиснути кнопку “Save”.

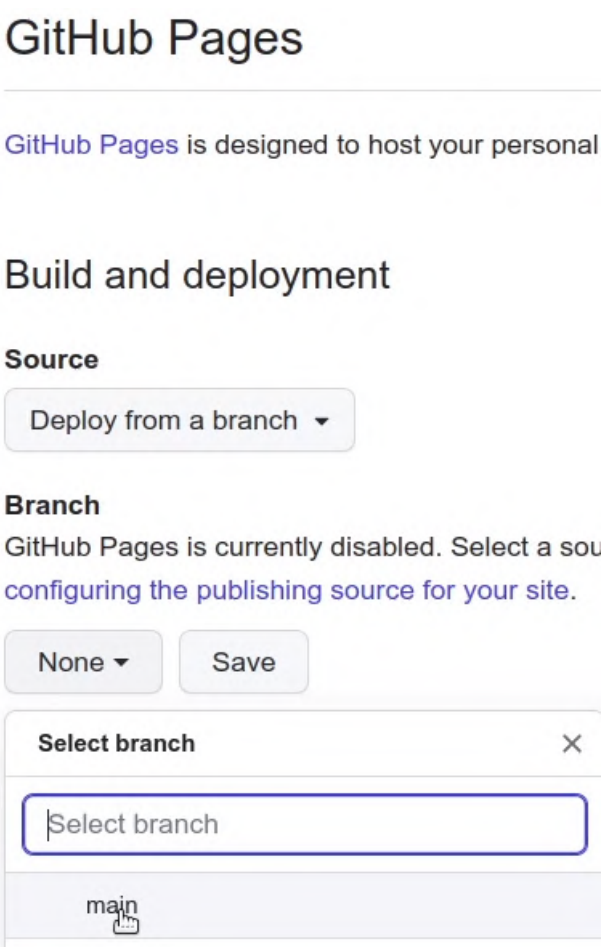

Рис. 1.10. Налаштування GitHub Pages.

4. Додати вміст до репозиторію (рис. 1.11).

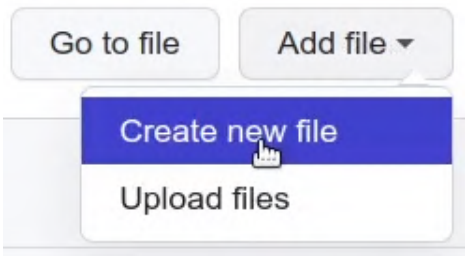

Рис. 1.11. Створення нового файлу у репозитарії.

5. Переглянути опублікований сайт за адресою `https://<username>.github.io/<repositoryname>` (рис. 1.12).

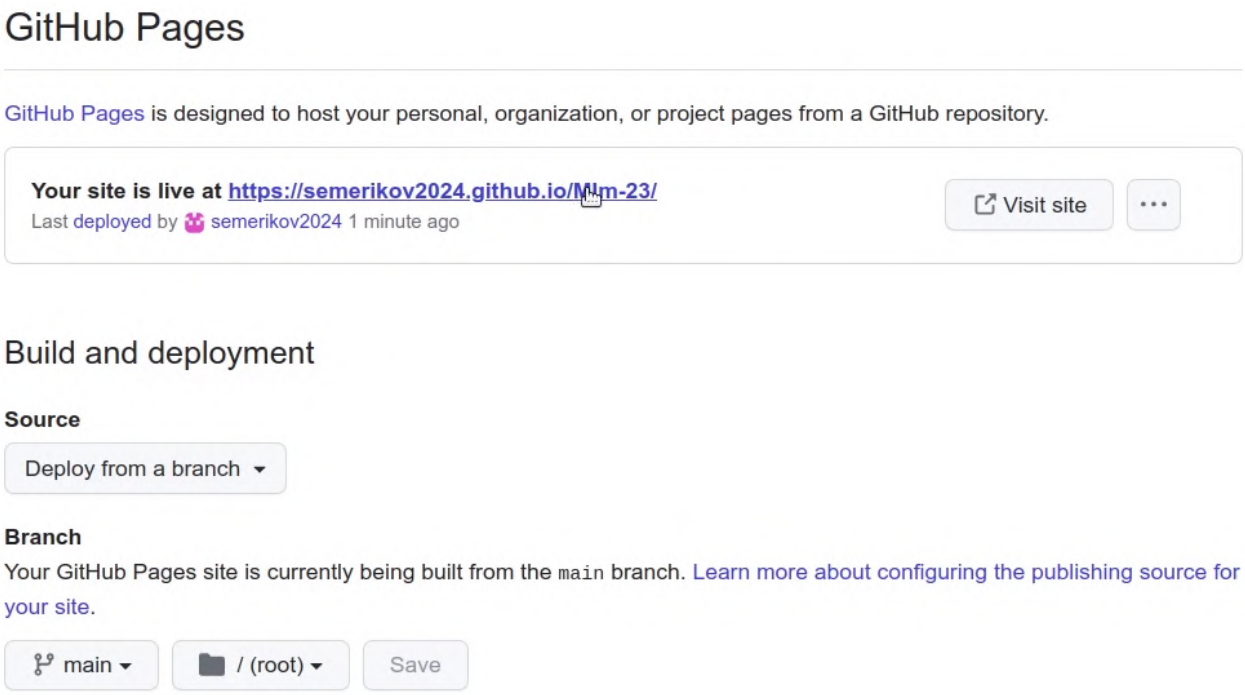

Рис. 1.12. Налаштування GitHub Pages.

У репозитарії GitHub кожен каталог повинен мати принаймні один файл, тому для створення каталогу необхідно обрати “Create new file” та увести ім’я файлу після імені каталогу, використовуючи “/” як стандартний роздільник (рис. 1.13).

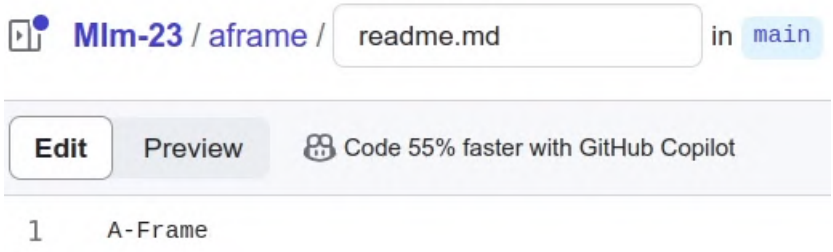

Рис. 1.13. Створення каталогу `aframe` разом із файлом `readme.md`.

Робота з репозитарієм GitHub через веб-інтерфейс не є найзручнішим варіантом, особливо за частих змін окремих файлів. Щоб покращити роботу з репозитаріями GitHub, можна використовувати сторонні інструменти та програми, такі як GitKraken (`https://www.gitkraken.com/`), GitHub CLI (`https://cli.github.com/`) та GitHub Desktop (`https://desktop.`

`github.com/`) – останній є рекомендованим для новачків. Ці інструменти пропонують більш широкий функціонал і зручний інтерфейс, ніж веб-інтерфейс GitHub, та доступні в автономному режимі, що дозволяє працювати з репозитаріями GitHub навіть при поганому інтернет-з'єднанні.

2 Вступ до доповненої реальності

2.1 Доповнена та віртуальна реальність

Під AR (augmented reality – доповнена реальність) будемо розуміти здатність пристрою, зокрема мобільного пристрою або веб-браузера, відстежувати зображення або відображати 3D-об'єкт поверх цього зображення. Головна ідея AR полягає в тому, щоб відобразити комп'ютерну модель у реальному часі та реальному просторі з метою взаємодії між користувачем у реальному просторі та 3D-моделі у віртуальному.

AR може бути як маркерним, так й безмаркерним. У маркерній AR пристрій відстежує 2D-маркер: коли він знаходиться, на ньому фактично відображається 3D-об'єкт. У безмаркерному варіанті пристрій буде шукати плоску поверхню (стіл, підлогу тощо), і розташовуватимемо 3D-об'єкт на ній.

Використовуючи камеру пристрою, AR надає можливості відображення комп'ютерно згенерованих об'єктів в ігрових, маркетингових та інших програмах – наприклад, для розстановки меблів у вітальні або примірки одягу перед їх покупкою. Це дійсно велика можливість для бізнесу – показати, як виглядає продукція, перш ніж будь-який споживач дійсно її купує. Для AR розробляються спеціальні пристрої, як правило, у вигляді шоломів та гарнітур, що надають можливість занурення користувача у модельне середовище.

AR доповнює реальний світ 3D-моделями, якими можна керувати за допомогою мобільного пристрою в будь-якому місці. Віртуальна реальність (Virtual Reality – VR) занурює користувача у модельний світ, для чого, як правило, необхідні наголовні дисплеї (Head Mounted Devices – HMD).

Інтерактивність у програмах для AR і VR забезпечується дуже схоже. Так, наприклад, VR фактично використовує контролери, а у деяких випадках й відстеження рук, що надає змогу користувачеві взаємодіяти з 3D-об'єктами всередині сцени, у якій вони знаходяться.

До головних небезпек використання HMD для роботи у VR відносяться:

- напруження очей;
- запаморочення і головні болі після використання HMD.

На відміну від VR, AR не має таких значних ризиків для здоров'я. Тим не менш, викликає занепокоєння можливість користувачів залишатися зосередженими на тому, що вони роблять, під час використання AR – зокрема, з причин безпеки.

Найбільш поширений тип пристроїв, готових для AR – смартфони та планшети з операційними системами iOS (версія 11 та вище під управлінням iPhone та iPad) та Android (версія 7 та вище).

HoloLens є HMD-подібною гарнітурою для AR, що знаходиться у активній розробці. Цю гарнітуру часто порівнюють з AR-окулярами, такими як CastAR, Meta, Laster SeeThru та K-Glass.

Для веб-браузерів AR доступна, якщо у них реалізована підтримка WebRTC та WebGL – насамперед Google Chrome та Mozilla Firefox.

2.2 Основи роботи у Three.js

WebGL (`https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API`) – JavaScript API для рендеринга 3D-графіки в браузерах. Він є крос-платформним стандартом відображення, який підтримують всі основні браузери. Проте низькорівневий код WebGL складно читати та писати, тому були створені більш зручні для користувача бібліотеки.

Three.js (`https://threejs.org/`) є однією з таких бібліотек, яку ми будемо використовувати. Її автор Рікардо Мігель Кабелло, також відомий як mrdoob, є одним із піонерів використання WebGL, тому ця бібліотека часто використовується при побудові інших бібліотек. Більшість WebAR SDK підтримують Three.js, тому вона дійсно потребує опанування для ефективної розробки веб-додатків з доповненою реальністю.

Щоб зрозуміти, як на високому рівні працює Three.js, доцільно провести аналогію з роботою фото- чи кінорежисеру, який:

1) налаштовує сцену шляхом розташування на ній об'єктів;

2) рухає камеру, щоб зафіксувати кадри з різних позицій та ракурсів (рис. 2.1).

Three.js не є спеціалізованою бібліотекою для доповненої реальності – вона містить суттєво більше функціональності, в тому числі тієї, що є більш придатною для веб-VR (освітлення, камери та ін.) (рис. 2.2).

Рис. 2.1. Фото Alex Simpson (`https://unsplash.com/@m_simpsan`).

Для створення додатку із застосуванням Three.js необхідно створити сторінку HTML, особливістю якою буде відсутність тіла, адже для генерації її вмісту буде використовуватись код JavaScript, що зазвичай розташовується між тегами `<script>` та `</script>`. Проте це не найкраща практика, тому доцільним є розміщення коду в окремому файлі main.js:

```
<html>
  <head>
    <script async src=
"https://unpkg.com/es-module-shims@1.8.0/dist/es-module-shims.js"
    ></script>
    <script type="importmap"> {
      "imports": {
        "three": "https://unpkg.com/three/build/three.module.js",
        "three/addons/": "https://unpkg.com/three/examples/jsm/"
      }
    }
    </script>
```

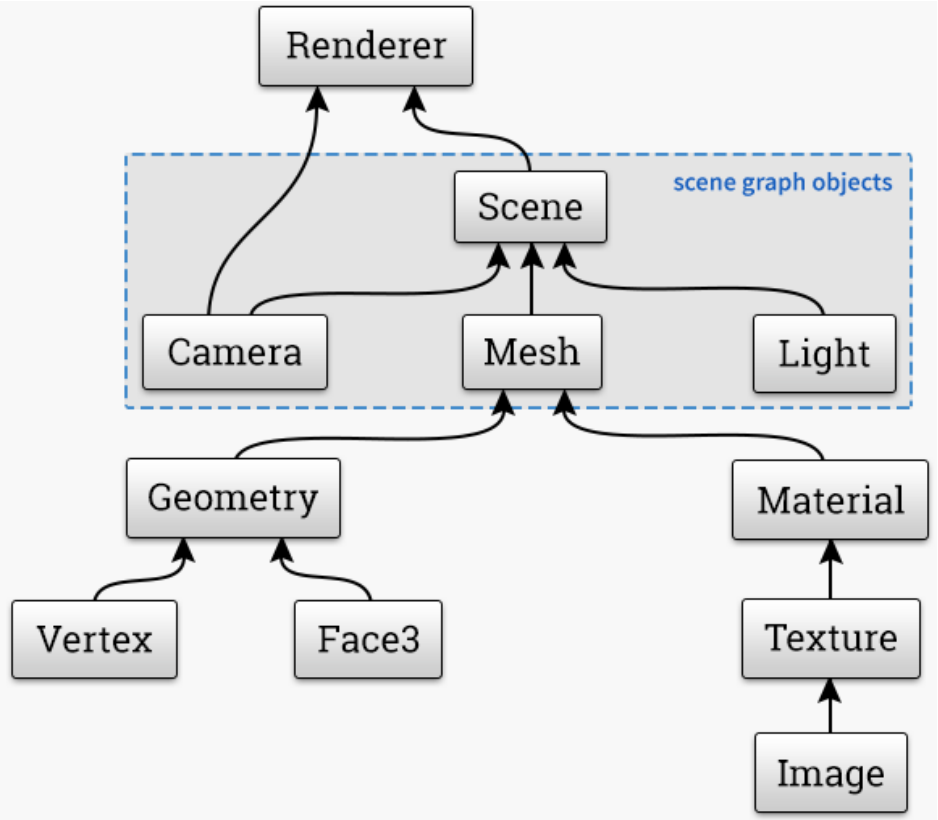

Рис. 2.2. Загальна структура Three.js.

```
    <script src="./main.js" type="module"></script>
  </head>
  <body>
  </body>
</html>
```

Бібліотека `es-module-shims` надає можливість підтримки модулів ES6 (ECMAScript 2015 – мова, розширенням якої є мова JavaScript) для браузерів, які ще не повністю підтримують цей стандарт.

Перше, що необхідно зробити, це імпортувати 3D бібліотеку. Наразі багато бібліотек, розміщених у хмарі, можна імпортувати безпосередньо через механізм CDN (content delivery network – мережу доставляння контенту), таких, як unpkg (`https://unpkg.com/`). На сторінці документації Three.js (`https://threejs.org/docs/index.html#manual/en/introduction/Installation`) подано відповідні приклади того, як це робити.

Так, при використанні звернення до CDN за шляхом `https://unpkg.com/three/` буде отриманий повний перелік версій Three.js, доступних че-

рез CDN. У випадку, коли необхідно зафіксувати номер версії бібліотеки (адже її функціональність змінюється – не лише з'являються нові можливості, а й зникають застарілі), доцільно чітко його вказати: наприклад, замість узагальненого `https://unpkg.com/three/build/three.module.js` вказати `https://unpkg.com/three@0.156.1/build/three.module.js`.

Конкретний шлях визначається у карті імпорту, що задається у `<script type="importmap">` парами виду `символ: шлях`. Перший символ – `three` – пов'язується із шляхом до модуля: библиотеки Three.js, що надає можливість її імпортувати із CDN:

```
import * as THREE from "three";
```

Намагання переглянути цей документ HTML локально у більшості сучасних браузерів завершиться невдачею через заборону локального (за протоколом `file://`) завантаження модулів, тому необхідним є застосування локального чи віддаленого веб-серверу.

Зазвичай код JavaScript має виконуватись після завершення завантаження документу HTML. Для того, щоб убезпечити це, слід додати обробник події `DOMContentLoaded`, усередині якого розташувати код:

```
document.addEventListener("DOMContentLoaded", () => {
  const scene = new THREE.Scene();

  const geometry = new THREE.BoxGeometry(1, 1, 1);
  const material = new THREE.MeshBasicMaterial({color: "#0000FF"});
  const cube = new THREE.Mesh(geometry, material);
  cube.position.set(0, 0, -2);
  cube.rotation.set(0, Math.PI/4, 0);
  scene.add(cube);

  const camera = new THREE.PerspectiveCamera();
  camera.position.set(1, 1, 5);

  const renderer = new THREE.WebGLRenderer({alpha: true});
  renderer.setSize(500, 500);
  renderer.render(scene, camera);
```

```
  document.body.appendChild(renderer.domElement);
});
```

Як показано на рис. 2.2, основою є сцена – для її створення необхідно викликати конструктор без параметрів класу `Scene` (змінна `scene` – об'єкту класу `Scene` створюється динамічно за допомогою виклику `new`).

Створення об'єктів у Three.js відбувається у три кроки:

1) визначення геометрії об'єкту – векторів позиції, кольорів та ін.: так, `BoxGeometry` відповідає за прямокутний паралелепіпед;

2) визначення матеріалу – способу рендерингу об'єкту (його оптичні властивості – колір, фактура, блиск тошо): так, `MeshBasicMaterial` відповідає матеріалу, що має власний колір і не відбиває промені;

3) композиція геометрії та матеріалу виконується за допомогою `Mesh`.

Створений у такий спосіб куб буде розташований у початку координат. Для зміни його позиції скористаємось властивістю `position`, успадкованою класом `Mesh` від свого батька – `Object3D`. Дана властивість є об'єктом класу `Vector3`, а `set` – його методом.

Аналогічно, властивість `rotation` зберігає кути нахилу об'єкту в радіанах. До речі, документація Three.js (див., наприклад, `https://threejs.org/docs/index.html#api/en/geometries/BoxGeometry`) містить інтерактивні демонстрації, що надають можливість переглянути різні об'єкти та модифікувати їх параметри. Рікардо Кабелло надає можливість конструювання сцени за допомогою візуального редактора за посиланням `https://threejs.org/editor/` – це може суттєво прискорити та полегшити процес її створення.

Всі об'єкти розміщуються на сцені за допомогою методу `add`.

Наступний об'єкт, що створюється – перспективна камера (`PerspectiveCamera`). Параметрами конструктора `PerspectiveCamera` є складові зрізаної піраміди огляду: перший – вертикальне поле зору (кут у градусах), другий – співвідношення сторін камери, третій – найближча площина, четвертий – найдальша (рис. 2.3).

Для зміни позиції камери скористаємось її властивістю `position`.

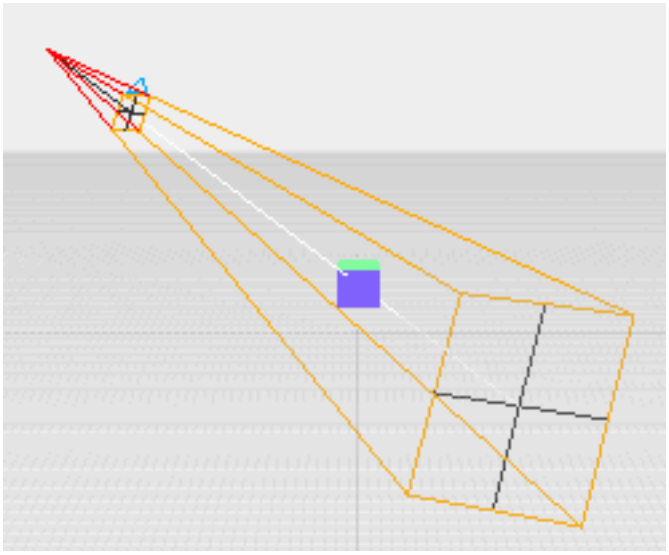

Рис. 2.3. Зрізана піраміда огляду.

Рендерер – це те, що буде відображати 3D-модель на полотні з урахуванням матеріалу, текстури та освітлення. Змінна `renderer` створюється динамічно за допомогою виклику `new` як об'єкт класу `WebGLRenderer`. Однойменна функція-конструктор класу в якості параметру приймає об'єкт у форматі JSON. Для роботи WebAR додатків важливо, щоб сцена була прозорою – тоді на неї можна буде накласти відеопотік з камери. Це досягається встановленням значенням параметру `alpha` у `true`.

Повний перелік властивостей та методів класу `WebGLRenderer` доступні у документації; в якості прикладу використано метод `setSize` встановлення висоти та ширини полотна (canvas) – площини, на яку проєціюється сцена.

Безпосередньо рендеринг виконує метод `render`, який відображає проєкцію сцени на холст із точки зору камери.

Останній крок – зв'язування полотна зі сторінкою HTML – виконується викликом

```
document.body.appendChild(renderer.domElement);
```

Отже, ми використовуємо елемент `canvas` для відображення результатів рендерингу (рис. 2.4).

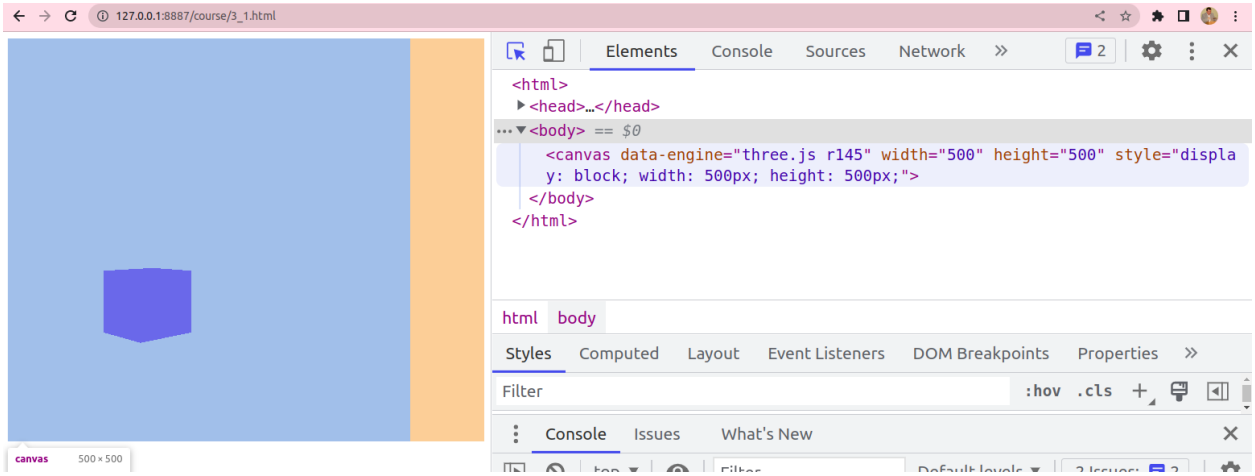

Рис. 2.4. Об'єкт `canvas`, убудований у документ HTML.

2.3 Перша програма у WebAR

Перед зв'язування полотна зі сторінкою HTML для WebAR додатків необхідно виконати підключення відеопотоку. Для цього до документу HTML додамо тег (об'єкт) `video`:

```
const video = document.createElement("video");
```

Усередині об'єкту `video` розмістимо відеопотік з камери:

```
navigator.mediaDevices.getUserMedia({video:true})
  .then((stream) => {
    video.srcObject = stream;
    video.play();
});
```

`Navigator` є інтерфейсом доступу до стану та властивостей певного веб-браузера. Отримати відповідний об'єкт можна зверненням до властивості вікна `window.navigator`. Метод `mediaDevices` повертає посилання на обєкт класу `MediaDevices`, що може бути використаний для отримання інформації про доступні медіапристрої (`enumerateDevices`), визначити їх властивості (`getSupportedConstraints`) та отримати доступ до них (`getUserMedia`).

Якщо виклик `getUserMedia` із запитом лише на відео (`video:true`) буде успішним, то буде отримано посилання на відеопотік `stream`, який необхідно пов'язати із властивістю `srcObject` об'єкту `video`. Останнім викли-

кається метод `play` для початку відтворення (програвання) відеопотоку з камери у об'єкті `video`.

Налаштуємо елементи каскадної таблиці стилів (CSS) для об'єктів `video` та `renderer`:

```
video.style.position = "absolute";
video.style.width = renderer.domElement.width;
video.style.height = renderer.domElement.height;
renderer.domElement.style.position = "absolute";
```

Значення параметру `position` встановлюється у `absolute` – такий елемент "щезає" з того місця, де він мав бути й позиціонується заново. Усі інші елементи розташовуються так, нібито цього елементу ніколи не було, а ширина елементу встановлюється за його вмістом. У нашому випадку ширина та висота об'єкту `video` встановлені у аналогічні значення полотна, на якому працює `renderer`, тому застосування `position:absolute` як до `video`, так й до `renderer` надає можливість сумістити (накласти) ці два об'єкти.

Додати створений та налаштований обєкт `video` до документу необхідно **до** додавання `renderer.domElement` – тоді зображення куба буде поверх відеопотоку:

```
document.body.appendChild(video);
```

На рис. 2.5 показано першу реалізацію WebAR, в якій реальний об'єкт з камери доповнений віртуальним об'єктом.

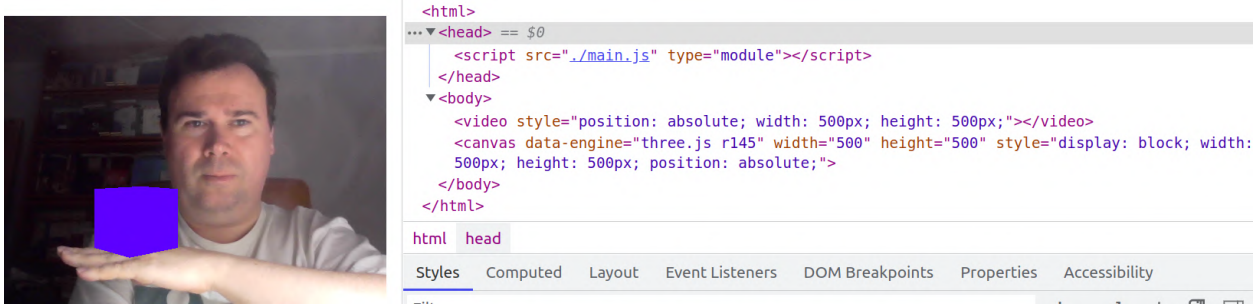

Рис. 2.5. Результат накладання.

Розміщення полотна поверх відео є основою WebAR. Єдине, чого бракує, це відображення об'єкту у більш доцільному місці та оновлення його положення відповідно до сигналу з камери, тобто відстеження об'єкту.

2.4 Відстеження та доповнена реальність

Змінити положення зображення можна шляхом переміщення віртуальної камери, змінюючи її позицію (координати) та нахил. Доцільні зміни вимагають відстеження об'єктів, тому поширеною є класифікація доповненої реальності на маркерну, безмаркерну, координатну тощо.

Автор бібліотеки MindAR пропонує класифікацію доповненої реальності за типом відстеження.

Перший тип – *відстеження зображень*: у цьому типі віртуальні об'єкти з'являються поверх цільових зображень, які можуть бути маркерними (barcode-like, рис. 2.6), які мають заздалегідь визначену структуру, та природними (рис. 2.7), які можуть бути чим завгодно.

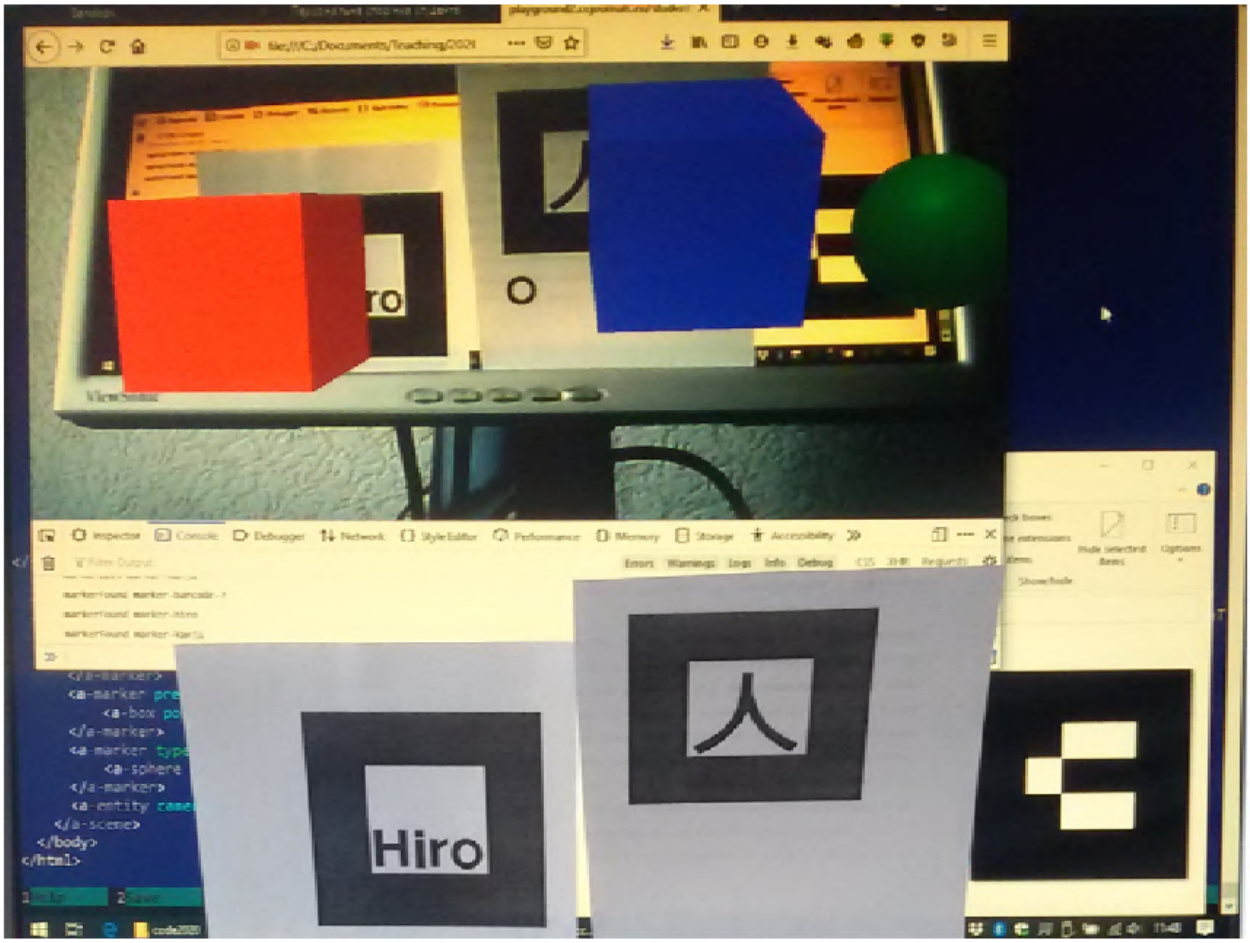

Рис. 2.6. Застосування маркерів.

Зображення не обов'язково маю бути друкованими чи екранними – можуть бути навіть футболки з доповненою реальністю[1].

[1]Див., наприклад, "9 ideas for creating tech-infused augmented reali-

Рис. 2.7. Застосування природних зображень.

Другий тип доповненої реальності – *відстеження обличчя*, за якого об'єкти прикріплюються до людського обличчя. Прикладами є фільтри в Instagram, Google Meet (рис. 2.8), кампанії у соціальних мережах, додатки для примірки віртуальних аксесуарів тощо.

Третій тип доповненої реальності – *відстеження довкілля* (world tracking), який також називають безмаркерною доповненою реальністю. За такого типу відстеження об'єкти доповненої реальності можуть бути розміщений де завгодно, не обмежуючись конкретним зображенням, обличчям або фізичними об'єктами.

Додатки відстеження довкілля безперервно фіксують і відстежують навколишнє середовище і оцінює фізичне положення користувача додатку. Найчастіше обєкти доповненої реальності прикріплюються до певної поверхні (рис. 2.9), зокрема, до землі.

Геокоординатна доповнена реальність (location-based AR), відома за Pokémon GO, Ingress тощо (рис. 2.10) передбачає прив'язку контенту до певного географічного положення – широти та довготи. Зазвичай ці програми відстежують довкілля, оскільки доповнений вміст, як правило, прикріплений до землі, а геокоординатна частина є скоріше додатковою умовою, виконання якої приводить до початку відстеження довкілля (або

ty T-shirts" від Ainars Klavins за посиланням `https://overlyapp.com/blog/9-ideas-for-creating-tech-infused-augmented-reality-t-shirts/`

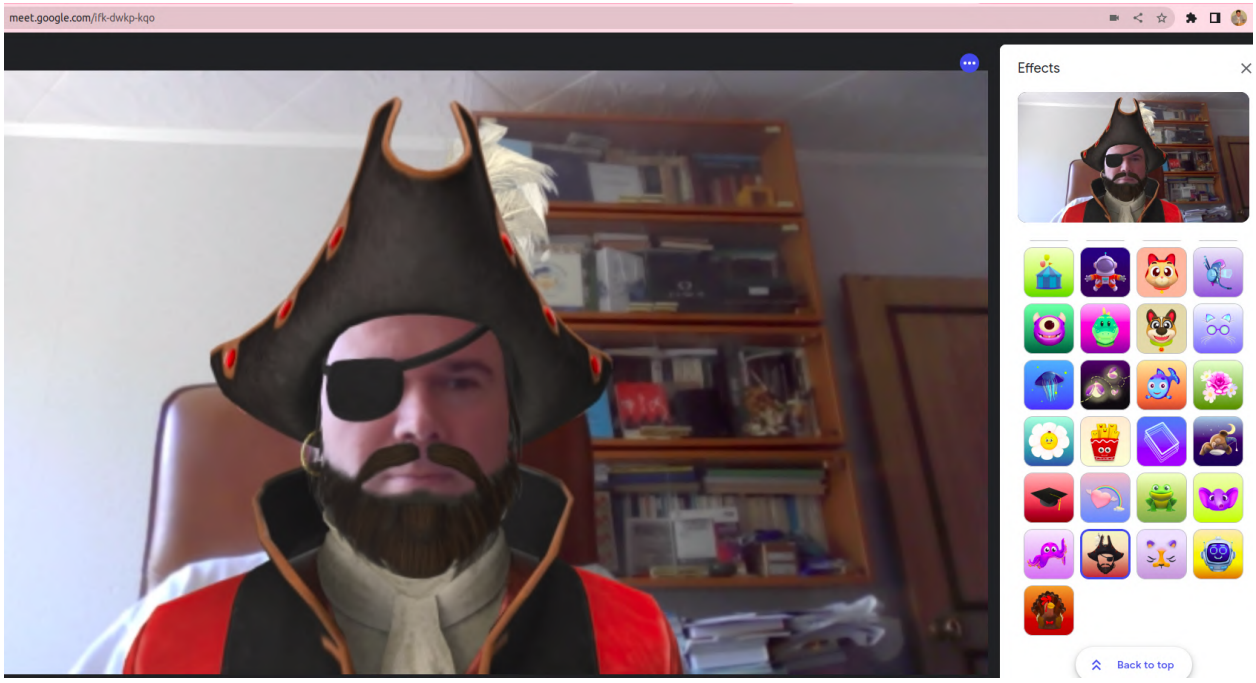

Рис. 2.8. Застосування фільтру для обличчя.

обличчя) у певному місці.

Можуть бути визначені й інші типи відстеження – відстеження 3D-об'єктів, відстеження рук та ін.

Незважаючи на різноманіття бібліотек для доповненої реальності, їх основною задачею є визначення позиції віртуальної камери відповідно до відстежуваного об'єкту, що ілюструється наступним псевдокодом:

```
const ar = new SOME_AR_ENGINE();
while(true)
{
  await nextVideoFrameReady();
  const {position, rotation} = ar.computeCameraPose(video);
  camera.position = position;
  camera.rotation = rotation;
}
```

Спочатку необхідно ініціювати бібліотеку – певний AR-рушій. Далі у безперервному циклі дочекатись кадр з відеопотоку реальної камери, визначити її положення (координати на нахил) та перемістити віртуальну камеру на полотні у те саме положення.

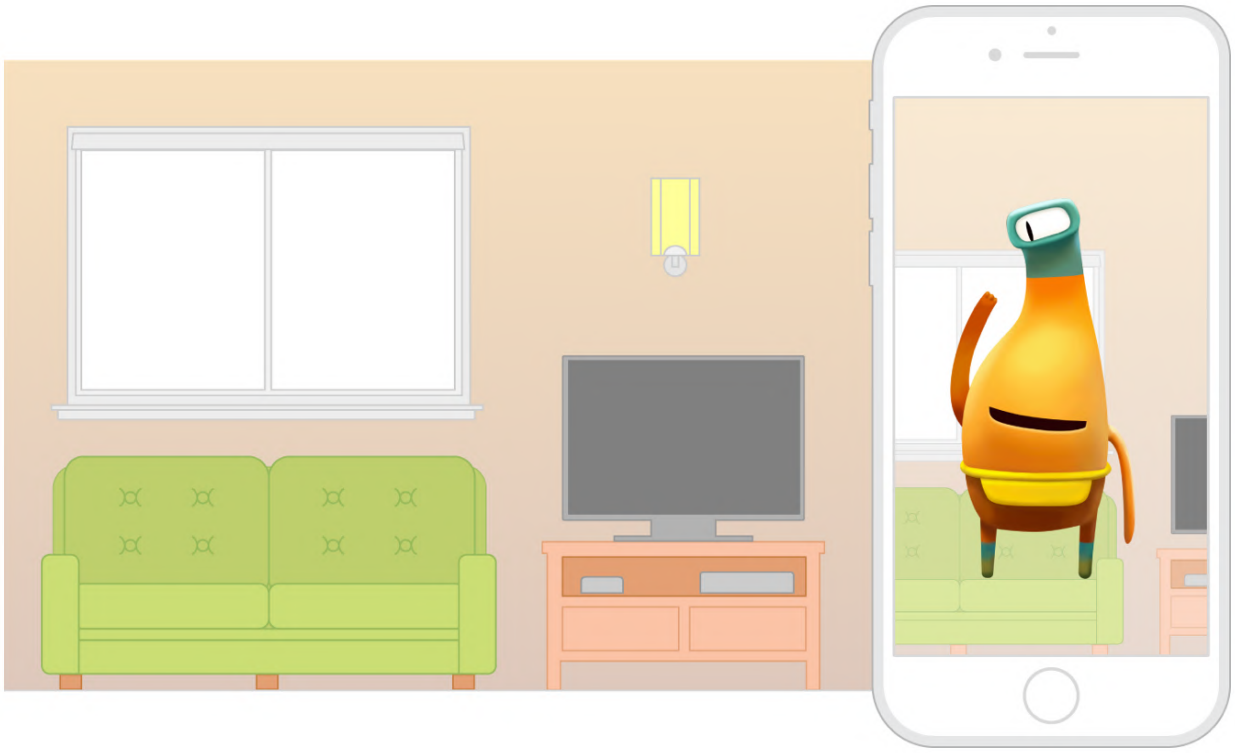

Рис. 2.9. Відстеження довкілля (`https://developer.apple.com`, Documentation / ARKit / Configuration Objects / Understanding World Tracking).

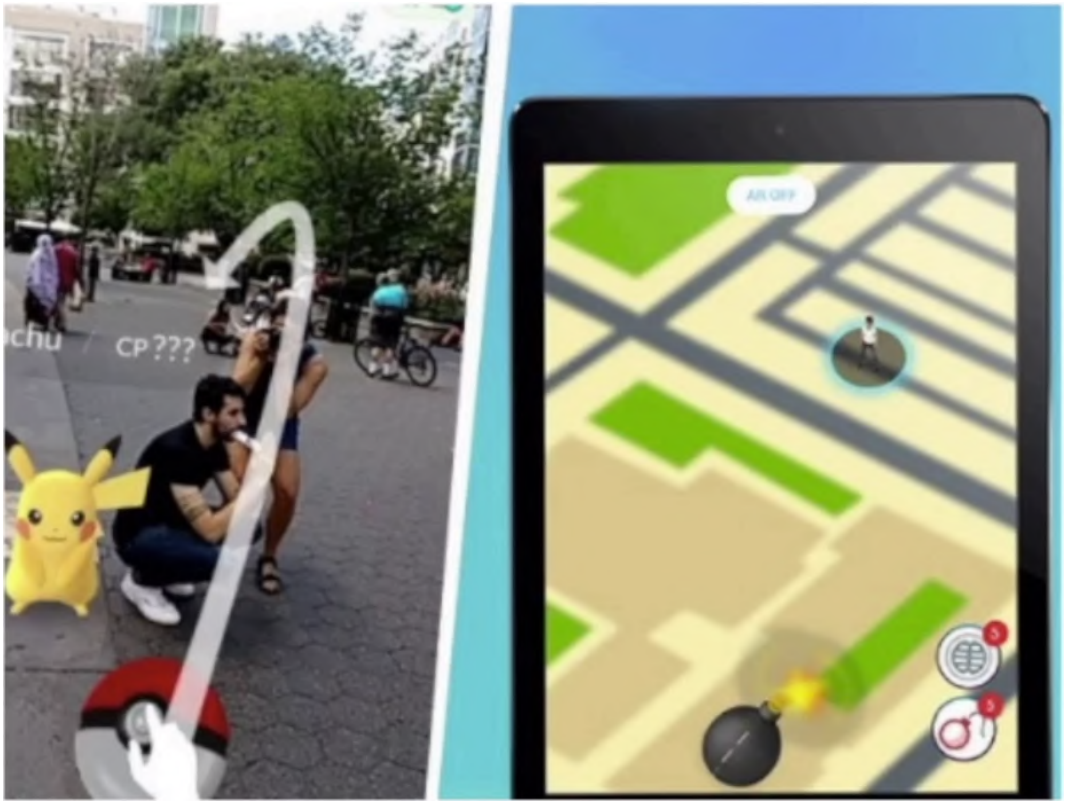

Рис. 2.10. Геокоординатна доповнена реальність.

Нерідко, однак, рухають не віртуальну камеру, а об'єкти на сцені – тоді визначається положення не реальної камери, а відстежуваного об'єкту, після чого об'єкт віртуальної реальності переміщується у те саме положення, що й відстежуваний об'єкт:

```
const ar = new SOME_AR_ENGINE();
while(true)
{
  await nextVideoFrameReady();
  const {position, rotation} = ar.computeObjectPose(video);
  cube.position = position;
  cube.rotation = rotation;
}
```

3 Розробка за допомогою A-Frame

3.1 Підключення бібліотеки A-Frame

A-Frame (`https://aframe.io`) дуже схожий на HTML – усі команди описуються тегами, які подібні до тегів HTML, але, на відміну від останніх, інтерпретуються не у веб-браузері на боці клієнта, а є способом доступу до JavaScript, що виконується на боці сервера. Разом із AR.js та MindAR він є потужним API для AR, що приховує деталі реалізації мовою JavaScript.

Гарний початок роботи із A-Frame пропонує A-Frame School (`https://aframe.io/aframe-school`).

Є декілька основних способів підключення A-Frame. Перший – застосувати збірку з офіційного сайту A-Frame, в якій конкретизується номер використовуваної версії:

```
<script src="https://aframe.io/releases/1.4.0/aframe.min.js">
</script>
```

За недоступності офіційного сайту можна використати одну з мереж поширення контенту – наприклад, jsDelivr із вказанням номер версії бібліотеки

```
<script src=
"https://cdn.jsdelivr.net/npm/aframe@1.4.2/dist/aframe.min.js">
</script>
```

чи без

```
<script
  src="https://cdn.jsdelivr.net/npm/aframe/dist/aframe.min.js">
</script>
```

Уточнити посилання на необхідну версію бібліотеки можна тут: `https://cdn.jsdelivr.net/npm/aframe/`. Поточна версія A-Frame доступна у репозитарії GitHub `https://github.com/aframevr/aframe/`, архів якого можна завантажити за посиланням `https://github.com/aframevr/aframe/archive/refs/heads/master.zip` – це буде корисним для локальної розробки.

Інший спосіб отримати власну копію бібліотеки – перейти за посиланням `https://aframe.io/docs/1.4.0/introduction/installation.html#include-the-js-build` та завантажити з нього файл збірки JavaScript (JS Build, Production Version – `https://aframe.io/releases/1.4.0/aframe.min.js`) (рис. 3.1).

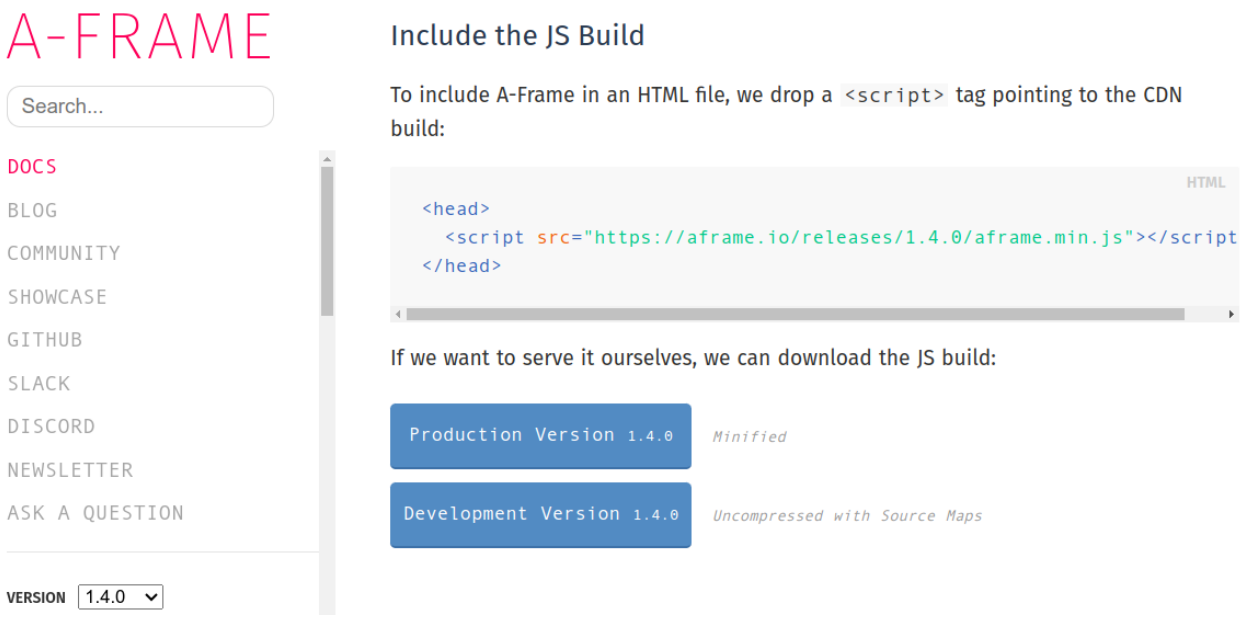

Рис. 3.1. Посилання на збірку A-Frame.

3.2 Створення сцени

Завантажений файл збірки необхідно зберегти у окремому каталозі (наприклад, `aframe`) всередині робочого каталогу, після чого створити у останньому індексний файл HTML `ARindex.html` із наступним вмістом:

```
<!DOCTYPE html>
<html>
  <head>
    <script src="aframe/aframe.min.js"></script>
  <head>
  <body>
    <a-scene>
      <a-sky color="grey"></a-sky>
    </a-scene>
  <body>
```

```
</html>
```

Файл містить команду `script` із посиланням на завантажений файл – вона завжди використовується для початку роботи з бібліотекою JavaScript. Саме у ньому інтерпретується тег `a-scene`, в якому знаходитиметься більша частина коду A-Frame. Тег `a-sky` створюватиме фоновий колір (його також можна застосувати для розміщення 360° зображення на сцені). Після відкриття файлу `ARindex.html` у веб-браузері отримаємо вікно із сірим фоном (рис. 3.2).

Рис. 3.2. Майже порожня сцена A-Frame.

Про те, що A-Frame API дійсно працює, свідчить режим VR (Virtual Reality) у нижньому правому куті сцени. Суттєво більше відомостей можна отримати, увімкнувши режим візуального 3D-інспектора (Ctrl-Alt-I) (рис. 3.3).

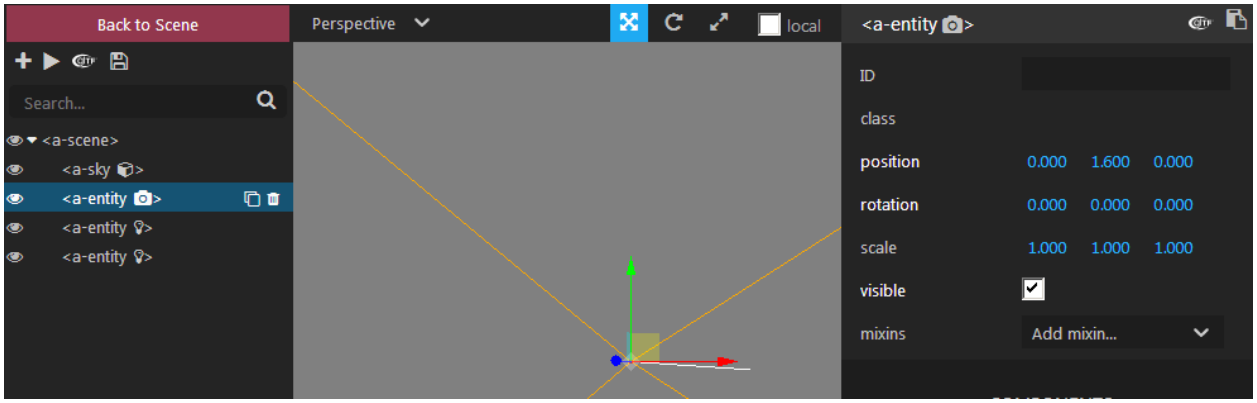

Рис. 3.3. Майже порожня сцена у інспекторі A-Frame.

Додавши до індексного файлу команду

```
<a-torus position="-2 1 -5" color="green" radius="1.2"></a-torus>
```

отримаємо зображення зеленого тору на сірому тлі (рис. 3.4).

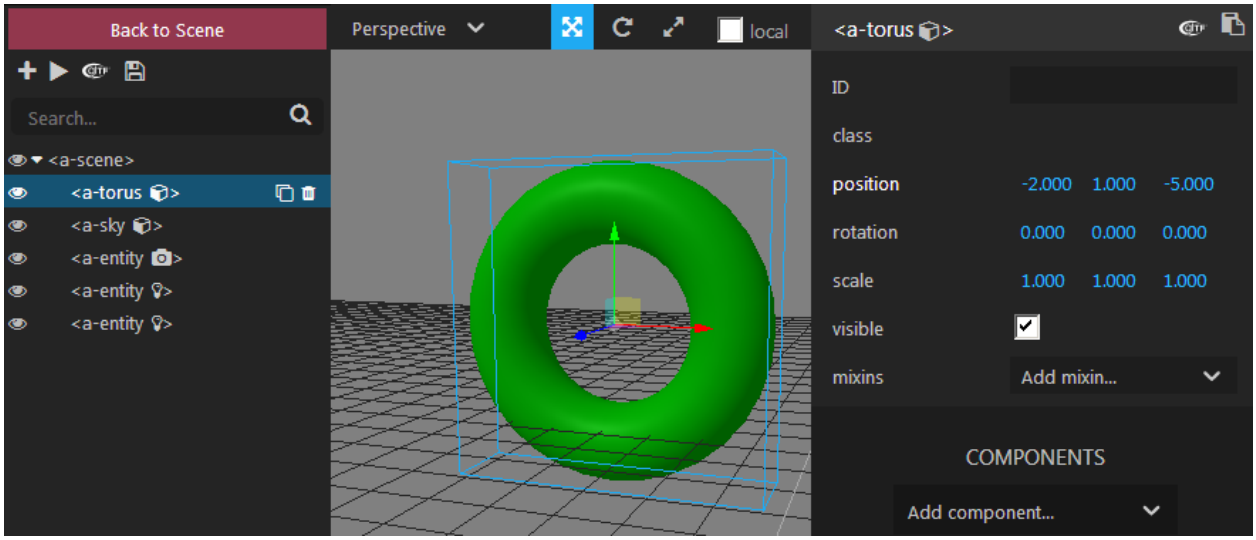

Рис. 3.4. Сцена із тором у інспекторі A-Frame.

У новостворених об'єктів є певні атрибути – ознайомитись із ними можна у документації до A-Frame.

3.3 Основні примітиви

Версія 1.4.2 підтримує наступні примітиви:

`a-box` – прямокутний паралелепіпед;

`a-camera` – камера (визначає, що бачить користувач);

`a-circle` – круг;

`a-cone` – конус;

`a-cursor` – опрацювання подій від миши;

`a-curvedimage` – панорамне зображення;

`a-cylinder` – циліндричні поверхні;

`a-dodecahedron` – дванадцятигранник;

`a-gltf-model` – 3D-модель у форматі glTF (.gltf);

`a-icosahedron` – двадцятигранник;

`a-image` – пласке зображення;

`a-light` – джерела світла;

`a-link` – гіперпосилання;

`a-obj-model` – 3D-модель у форматі Wavefront (.obj/.mtl);

`a-octahedron` – восьмигранник;

`a-plane` – площина;

`a-ring` – пласке кільце або диск;

`a-sky` – додає фоновий колір або 360° зображення;

`a-sound` – джерело звуку;

`a-sphere` – сфера або багатогранник;

`a-tetrahedron` – трикутна піраміда;

`a-text` – плаский текст;

`a-torus-knot` – тороподібна фігура;

`a-torus` – тор;

`a-triangle` – трикутна поверхня;

`a-video` – відео як текстура на площині;

`a-videosphere` – 360° фонове відео.

3.4 Атрибути об'єктів

Додамо до попередньої сцени, що містить зелений тор та сірий фон, ще кілька примітивів. Спочатку вставимо між тором та фоном площину:

```
<a-plane width="7" height="7" rotation="50 0 0"
         position="3 -2 -3" color="purple"></a-plane>
```

Для перегляду параметрів a-plane слід скористатись документацією за посиланням `https://aframe.io/docs/1.4.0/primitives/a-plane.html` – тут можна знайти різні атрибути площини, які можна застосувати до неї в тегу A-Frame. Параметри `width` та `height` відповідають за ширину та висоту прямокутника, `rotation` вказує на необхідність її повороту на 50° відносно вісі x та на 0 – відносно y та z, `position` – координати початку площини за відповідними осями, а `color` – обраний колір.

Площину, налаштовану у такий спосіб, на сцені можна побачити лише в режимі інспектора, тому що вона розміщена дуже низько. Якщо відсунути площину далі від камери зміною координат її початку на `-2 -2 -5`, її можна побачити (рис. 3.5).

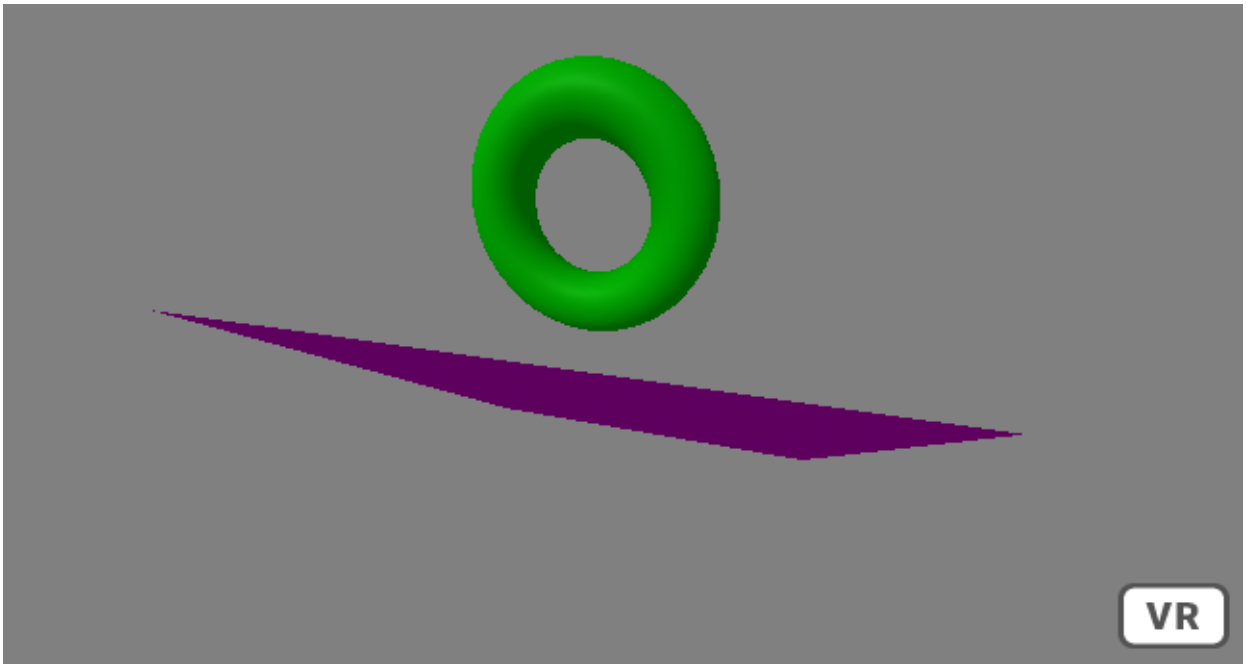

Рис. 3.5. Сцена із тором та площиною.

Тепер, коли у нас є площина, додамо ще кілька об'єктів. Створимо циліндричний об'єкт у формі льодяника жовтого кольору, розташованого під тором (рис. 3.6):

```
<a-cylinder color="yellow" height="2" radius="0.05"
            position="-2 -1 -5"></a-cylinder>
```

Додамо ще один циліндр (рис. 3.7):

```
<a-cylinder color="blue" height="2" radius="0.05"
            position="-3 -1 -5"></a-cylinder>
```

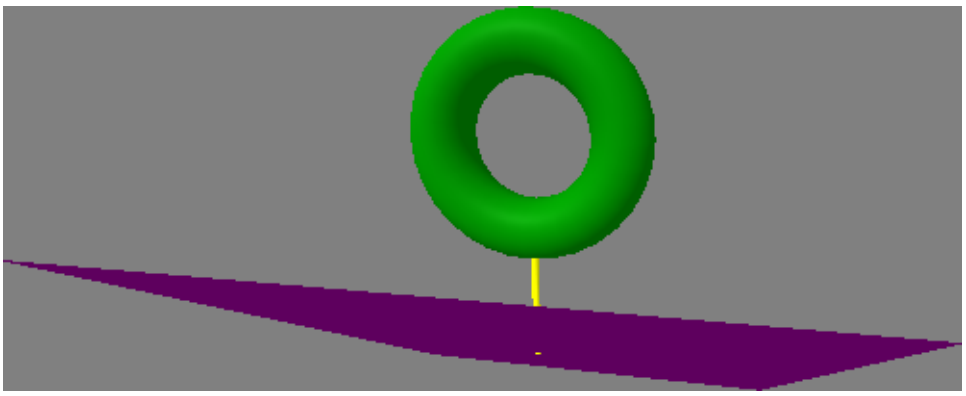

Рис. 3.6. Сцена із тором, площиною та циліндром.

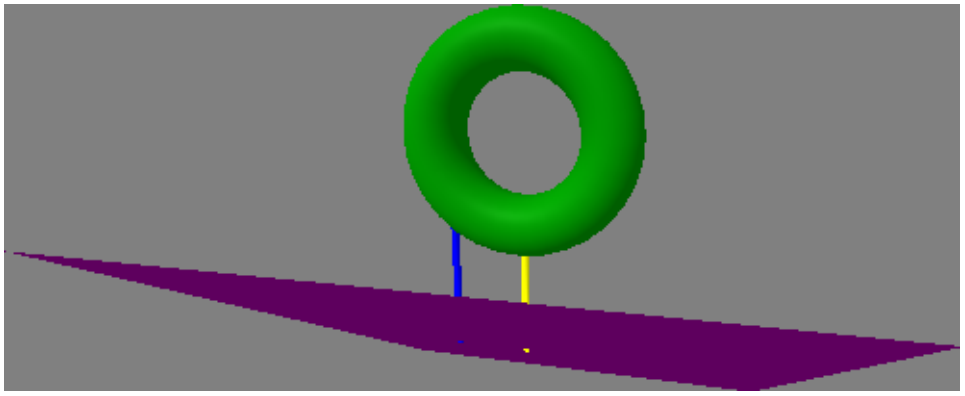

Рис. 3.7. Сцена із тором, площиною та двома циліндрами.

І, нарешті, тороподібну фігуру (рис. 3.8; досить цікава форма – без документації не розібратись):

```
<a-torus-knot color="orange" radius="1.2" position="-3 1 -5">
</a-torus-knot>
```

3.5 Додавання тексту

Виконаємо невеликі зміни у індексному файлі в атрибуті обертання площини – зміна кута огляду на `-55 0 0` надає їй суттєво кращого вигляду (рис. 3.9).

Додамо до сцени площину, перпендикулярну попередній:

```
<a-plane width="9" height="2" position="3 1 -9"></a-plane>
```

За замовчанням її колір буде білим – достатньо зручний для того, щоб перед ним розмістити напис (рис. 3.10):

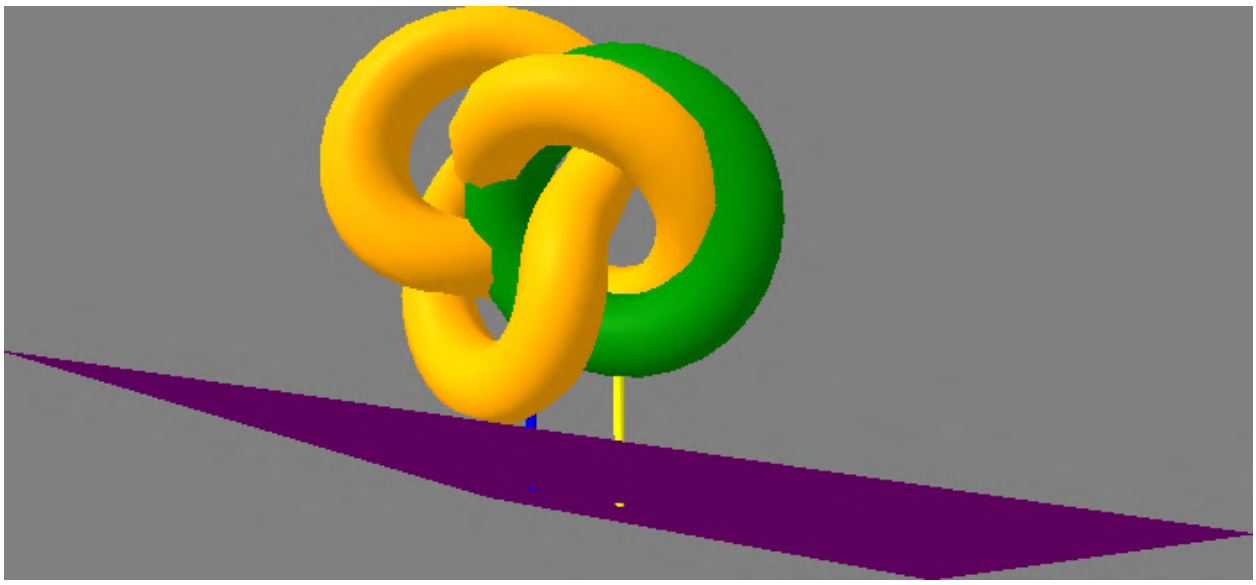

Рис. 3.8. Сцена із тором, площиною, двома циліндрами та тороподібною фігурою.

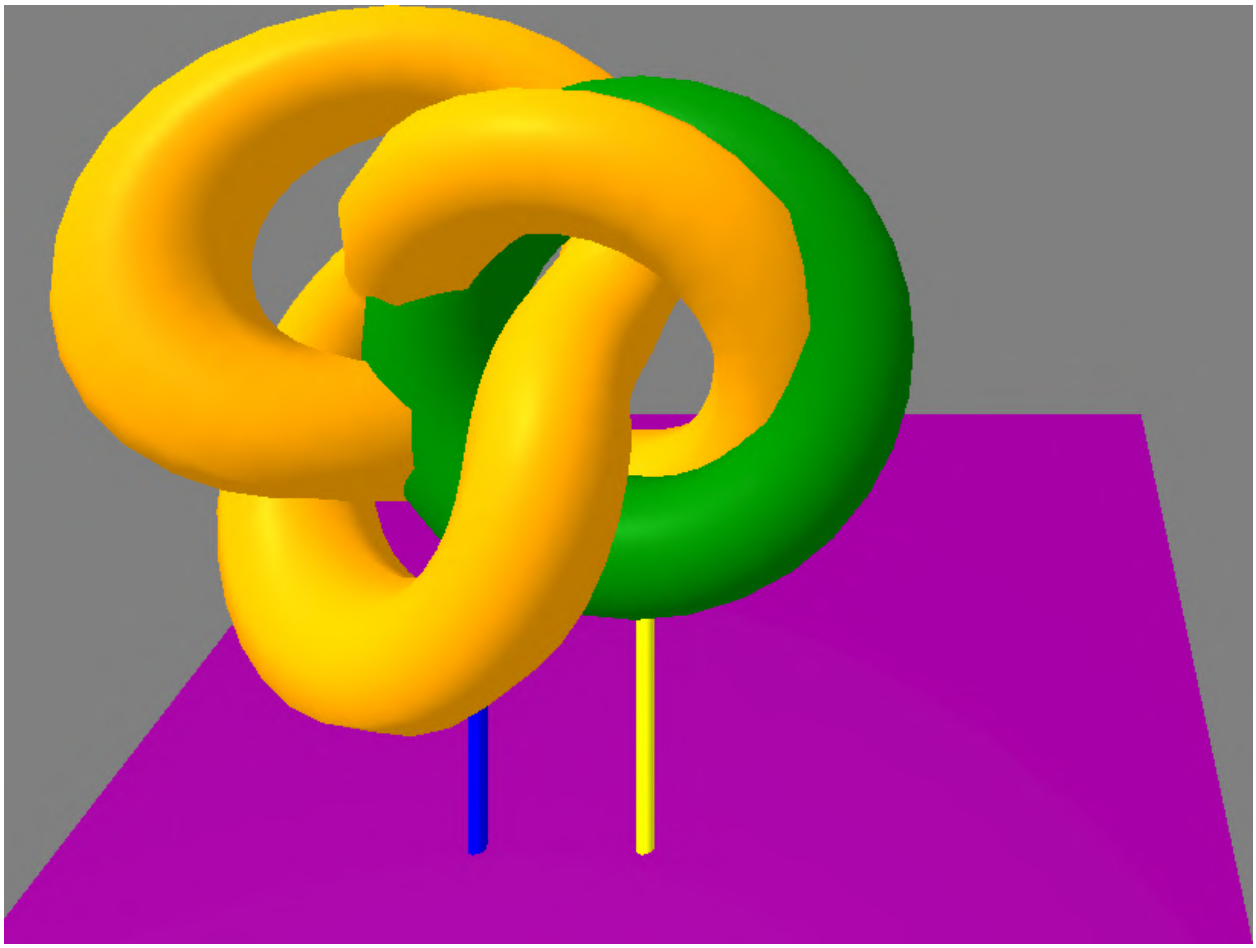

Рис. 3.9. Поворот площини на 105°.

```
<a-text value="Welcome to browser's VR!" color="black"
width="10" position="-0.5 1 -6"></a-text>
```

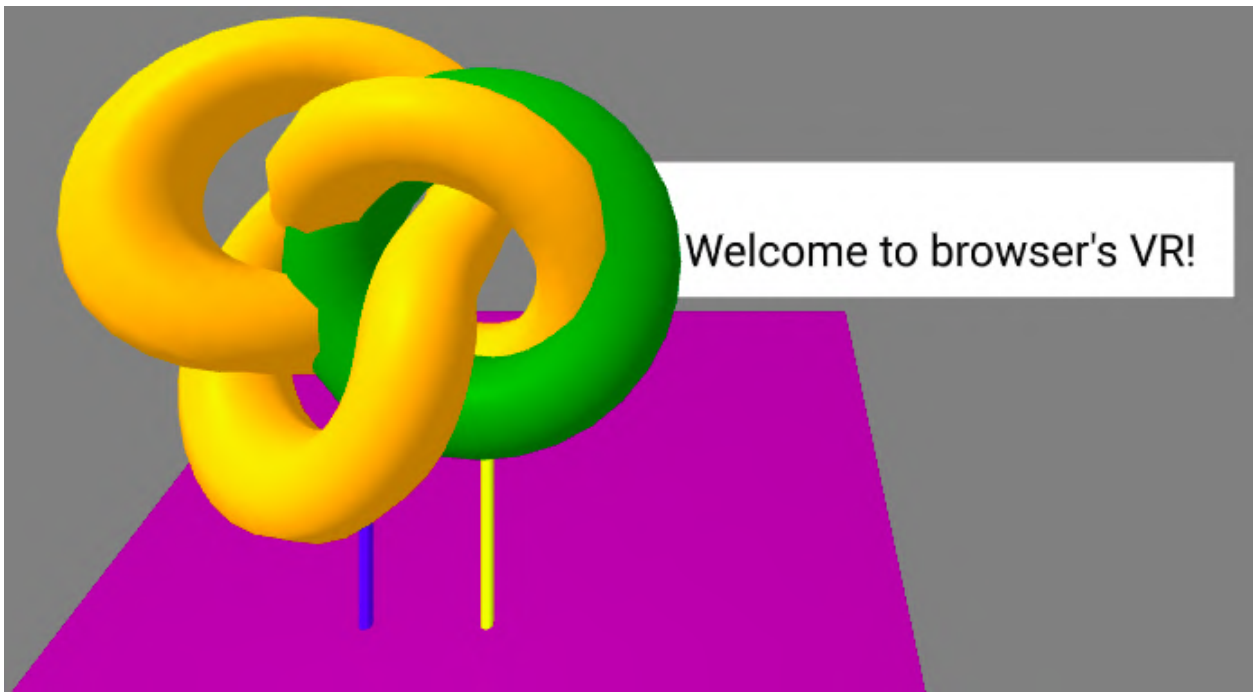

Рис. 3.10. Розміщення тексту на сцені.

3.6 Створення власних шрифтів

Лобова спроба замінити даний напис на кириличний, скоріше за все, виявиться невдалою без використання відповідного шрифта. Більш ніж 2000 різних шрифтів та способи їх використання можна знайти у репозитарії `https://github.com/etiennepinchon/aframe-fonts`. Будь-який шрифт із репозитарію може бути підключений за посиланням виду `https://raw.githubusercontent.com/etiennepinchon/aframe-fonts/master/fonts/[FONT_NAME]/[FONT_TYPE].json`, та чи багато серед них кирилічних?

Тому пропонуємо створити власний шрифт за допомогою MSDF font generator (`https://msdf-bmfont.donmccurdy.com/`). Для генерації можна застосувати пропонований шрифт Microsoft YaHei або завантажити інший шрифт у форматі True Type.

Архів, завантажуваний з MSDF font generator (рис. 3.11), містить шрифт у форматах JSON та PNG (для вихідного шрифту segoescb.ttf це будуть segoescb-msdf.json та segoescb.png відповідно – розташуйте їх там само, де знаходиться й ARindex.html). Чим більше символів обирається, тим більше має бути розмір PNG-файлу (до 1024x1024). Для економії трафіку

до них включені лише ті символи, які були обрані користувачем – всі інші не відображатимуться. Якщо користувач намагатиметься відобразити символ, відсутній у шрифті, він не відображатиметься або замість нього буде відображатись інший, тому не забудьте до переліку символів включити символ пробілу.

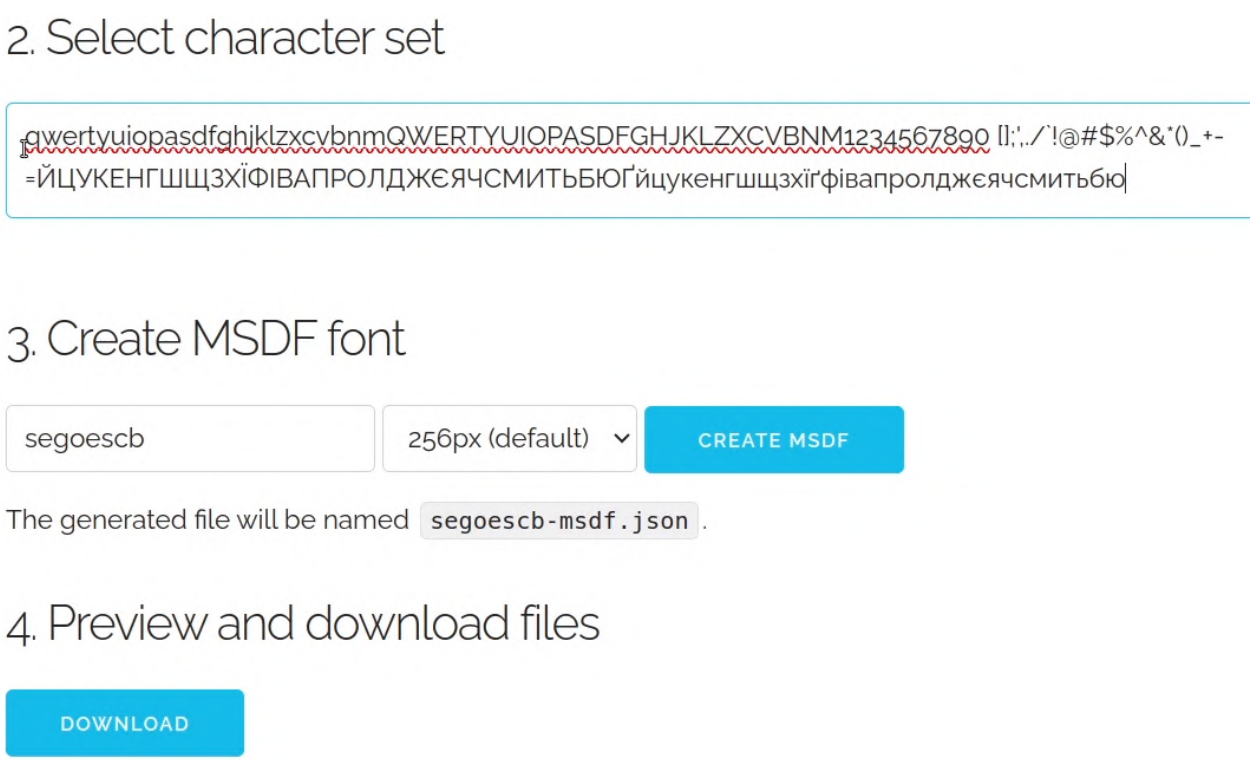

Рис. 3.11. Застосування генератора шрифтів.

Внесемо зміни до параметру `value` та додамо параметри `font` і `negate`:

```
<a-text value="Вітаємо у браузерній VR!" color="black" width="10"
position="-0.5 1 -6" font="segoescb-msdf.json" negate="false">
</a-text>
```

Якщо останній матиме істинне значення, обраний колір стане фоновим для кожної з літер (рис. 3.12).

Для завершення локалізації надпису доцільно масштабувати його до розміру площини, вказавши параметр `scale`.

Зауважимо, що застосування символів, які не входять до набору ASCII, потребує додаткового вказання одного з найпоширеніших кодувань тексту – UTF-8 – у заголовку документу:

```
<meta charset="utf-8">
```

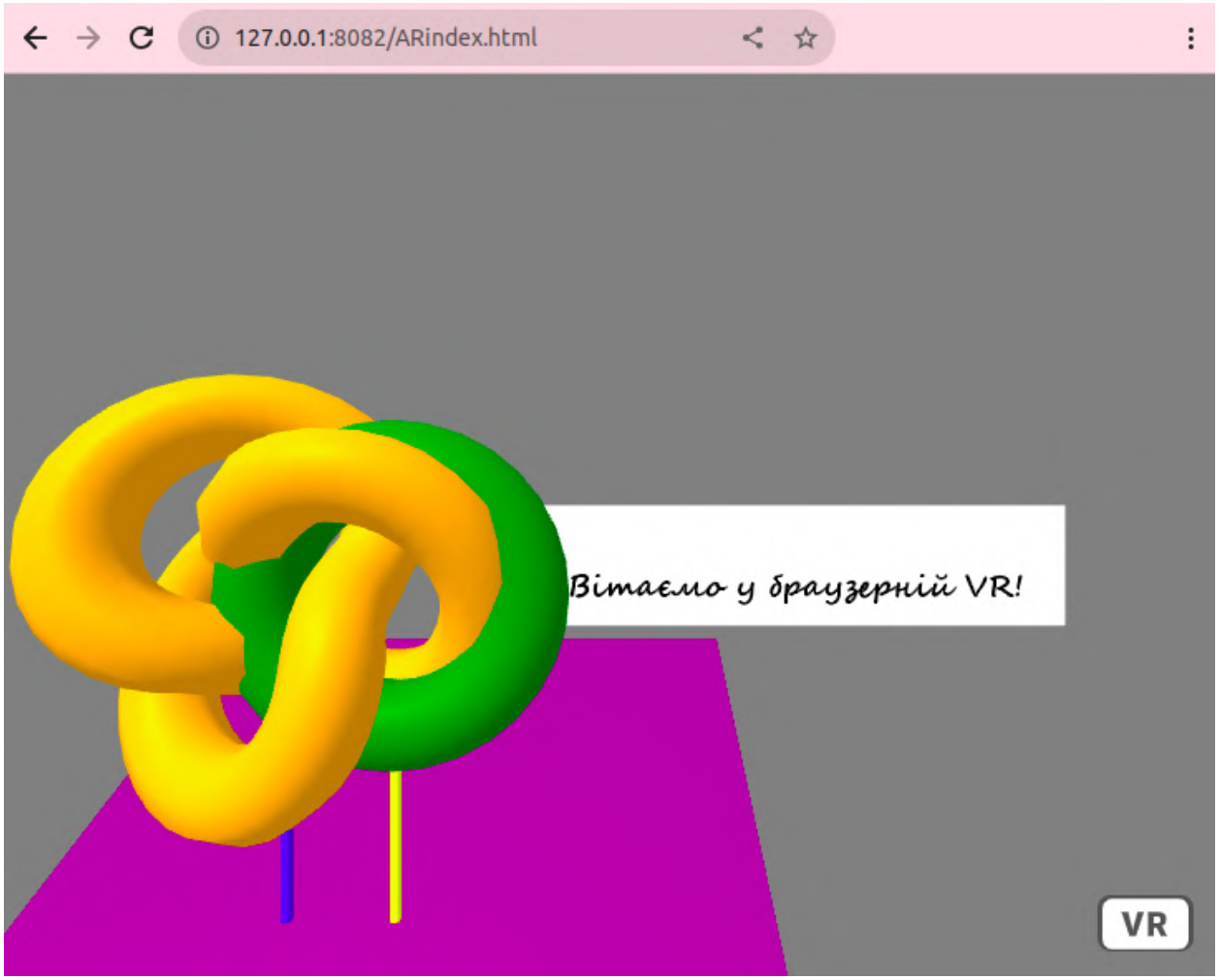

Рис. 3.12. Демонстрація згенерованого шрифту.

```
<meta name="viewport"
      content="width=device-width, initial-scale=1" />
```

Інший елемент `<meta>` використовується для налаштування метаданих для відображення веб-сторінки на мобільних пристроях:

- `width=device-width` тегу вказує браузеру використовувати фактичну ширину екрана пристрою (як в `device-width`) для відображення вмісту веб-сторінки – це робить її більш адаптивною до різних розмірів екрану;
- `initial-scale=1` задає початковий масштаб відображення сторінки: за значення 1 сторінка буде відображатися в натуральних розмірах, без змін масштабу, що надає можливість побачити весь її вміст без необхідності масштабування.

3.7 Анімація об'єктів

Для того, щоб анімувати примітив, необхідно додати до нього параметр `animation`:

```
<a-torus position="-2 1 -5" color="green" radius="1.2"
  animation="property: components.material.material.color;
    type: color; from: green; to: red; loop: true; dur: 10000">
</a-torus>
```

У даному прикладі основним в параметрі `animation` є `property` – та складова об'єкту, яка буде змінюватись (`components.material.material.color` – колір). `dur` визначає тривалість анімації в мілісекундах (10000 мс = 10 с), `from` задає початкове значення атрибуту, `to` – кінцеве, а `loop` визначає кількість повторень циклу зміни (`true` відповідає нескінченному циклу).

Наступний приклад демонструє зміну атрибуту `rotation` для того, щоб змусити тороподібний об'єкт обертатися навколо центру:

```
<a-torus-knot color="orange" radius="1.2" position="-3 1 -5"
animation="property:rotation; to: 0 0 360; loop:true; dur:10000">
</a-torus-knot>
```

Більш детально про параметр `animation` можна прочитати у довідці з A-Frame Core API за посиланням `https://aframe.io/docs/1.4.0/components/animation.html`.

3.8 Додавання текстур до об'єктів

Серед репозиторіїв A-Frame (`https://github.com/orgs/aframevr/repositories`) можна знайти assets та sample-assets – приклади файлів зображень, моделей та аудіозаписів для використання у A-Frame. Для того, щоб скористатися ними, необхідно отримати пряме (raw) посилання на об'єкт репозитарію, що розпочинається з `https://raw.githubusercontent.com/`, та додати його у якості атрибуту `src` до об'єкту, для якого обране зображення виступатиме текстурою. Змінимо площину під торами (рис. 3.13):

```
<a-plane src="https://raw.githubusercontent.com/aframevr/sample-
    assets/master/assets/images/illustration/758px-Canestra_di_
    frutta_(Caravaggio).jpg" width="7" height="7"
    rotation="-55 0 0" position="-2 -2 -5" color="purple">
</a-plane>
```

Рис. 3.13. Текстурована площина.

3.9 Управління ресурсами

Посилання на файл текстури може бути узяте із будь-якого місця, але не завжди сайти дозволяють використовувати прямі посилання на файли сайту поза його межами. “Політика того ж походження” (same origin policy) є важливим механізмом безпеки у сучасних веб-браузерах, що стосується як виконуваних у браузері файлах, так й використовуваних. Найчастіше вона реалізується через Cross-Origin Resource Sharing (CORS, спільне використання ресурсів з різних джерел) — механізм безпеки сучасних браузерів, який дозволяє вебсторінкам використовувати дані, що знаходяться в інших доменах.

Примітиви A-Frame можуть бути вкладені один в одного так само, як вони вкладені у сцену – це надає можливість відносної зміни координат,

кутів повороту тощо одного примітиву відносно іншого, їх групування, спільного управління тощо. Так, якщо розмістити усі об'єкти сцени між `<a-entity>` та `</a-entity>`, то ми отримаємо *модель*, додавання до якої параметрів `position`, `rotation`, `scale` та інших дозволить її переміщувати, повертати, масштабувати тощо.

A-Frame має систему керування ресурсами, яка попередньо завантажувати та кешувати їх для кращої продуктивності – це надає можливість уникнути затримок візуалізації, можливих при використанні прямих URL-посилань.

Ресурси розміщуються у блоці `<a-assets>...</a-assets>`, який, у свою чергу, розміщується на сцені. Типи ресурсів:

- `<audio>` – звукові файли (аудіо);
- `<img>` – текстури зображень;
- `<video>` – вудеотекстури;
- `<a-asset-item>` – інші ресурси, такі як 3D-моделі, шрифти та матеріали.

Сцена не буде відтворена або ініціалізована, доки браузер не отримає усі ресурси або система попереднього завантаження ресурсів не досягне тайм-ауту.

3.10 Завантаження моделей

A-Frame надає можливість застосовувати й сторонні моделі (`https://aframe.io/docs/1.4.0/introduction/faq.html#where-can-i-find-assets`) у різних форматах, найпоширенішими з яких є glTF (GL Transmission Format) та OBJ. Моделі у форматі OBJ складаються з геометрії (файл .OBJ) та матеріалів (файл .MTL) і можуть бути завантажені за допомогою примітиву `<a-obj-model>`, проте краще розділити дії із завантаження ресурсів (assets – моделей, текстур, відео, аудіо тощо) та їх візуалізації.

Переробимо сцену шляхом додавання до неї нових ресурсів та винесення вже наявних до блоку `<a-assets>...</a-assets>`:

```
<a-scene stats>
  <a-assets>
    <img id="canestra" src="https://raw.githubusercontent.com/
      aframevr/sample-assets/master/assets/images/illustration/
      758px-Canestra_di_frutta_(Caravaggio).jpg"
      crossorigin="anonymous">
    <img id="grid" src="https://raw.githubusercontent.com/
      aframevr/sample-assets/master/assets/images/uvgrid/
      UV_Grid_Sm.jpg" crossorigin="anonymous">
    <video id="city" src="https://cdn.aframe.io/
      360-video-boilerplate/video/city.mp4"
      crossorigin="anonymous"></video>
    <a-asset-item id="crate-obj"
      src="https://raw.githubusercontent.com/aframevr/assets/
      master/test-models/models/crate/crate.obj"></a-asset-item>
    <a-asset-item id="crate-mtl"
      src="https://raw.githubusercontent.com/aframevr/assets/
      master/test-models/models/crate/crate.mtl"></a-asset-item>
    <a-asset-item id="brainstem"
      src="https://raw.githubusercontent.com/aframevr/assets/
      master/test-models/models/glTF-2.0/brainstem/
      BrainStem.gltf"></a-asset-item>
  </a-assets>
  <a-sky color="grey"></a-sky>
  <a-torus position="-2 1 -5" color="green" radius="1.2"
    animation="property: components.material.material.color;
    type: color; from: green; to: red; loop: true; dur: 10000">
  </a-torus>
  <a-plane src="#canestra" width="7" height="7" rotation="-55 0 0"
    position="-2 -2 -5" color="purple"></a-plane>
  <a-cylinder color="yellow" height="2" radius="0.05"
    position="-2 -1 -5"></a-cylinder>
  <a-cylinder color="blue" height="2" radius="0.05"
    position="-3 -1 -5"></a-cylinder>
```

```
  <a-torus-knot color="orange" radius="1.2" position="-3 1 -5"
    animation="property: rotation; to: 0 0 360; loop: true;
    dur: 10000"></a-torus-knot>
  <a-plane src="#grid" width="9" height="2" position="3 1 -9">
  </a-plane>
  <a-text value="Вітаємо у браузерній VR!" color="black"
    width="10" position="-0.5 1 -6" font="segoescb-msdf.json"
    negate="false"></a-text>
  <a-videosphere src="#city"></a-videosphere>
  <a-obj-model src="#crate-obj" mtl="#crate-mtl"
    position="2.5 -0.5 -5"></a-obj-model>
  <a-gltf-model src="#brainstem" position="2 0.5 -5"></a-gltf-model>
  <a-light type="point" intensity="1.5" position="4 0 0"></a-light>
</a-scene>
```

Для використання зображень (`<img>`), відео (`<video>`) та моделей (`<a-asset-item>`), завантажених у блоці ресурсів, застосовуються селектори – звернення до ідентифікаторів ресурсів, яким передує знак #. Так, `#city` є селектором відеотекстури, яка накладається на сферу `<a-videosphere>` та перекриває `<a-sky>`. Якщо програма розпочинає виконуватись із спроби автоматичного програвання відео, вона навряд чи буде вдалою – більшість веб-браузерів дозволяють програвання відео та аудіо лише після певної дії користувача.

`<a-light>` створює точкове джерело світла з інтенсивністю 1.5, розташоване у вказаних координатах – вони використовується для освітлення розташованої поряд моделі у форматі OBJ (`<a-obj-model>`).

Ми рекомендуємо використовувати моделі у форматі glTF (`<a-gltf-model>`), якщо це можливо, оскільки glTF стає стандартом для передавання 3D-моделей через Інтернет. Моделі у форматі glTF являють собою множину файлів, для яких текстовий файл .gltf є точкою входу. Більш зручними для використання може виявитись бінарний файл .glb, що є архівом усіх файлів, необхідних для подання моделі.

A-Frame 1.4.0 не надає можливості відтворення вбудованої анімації моделі, проте це можна зробити за допомогою додаткового компоненту A-Frame (`https://github.com/c-frame/aframe-extras/tree/master`)

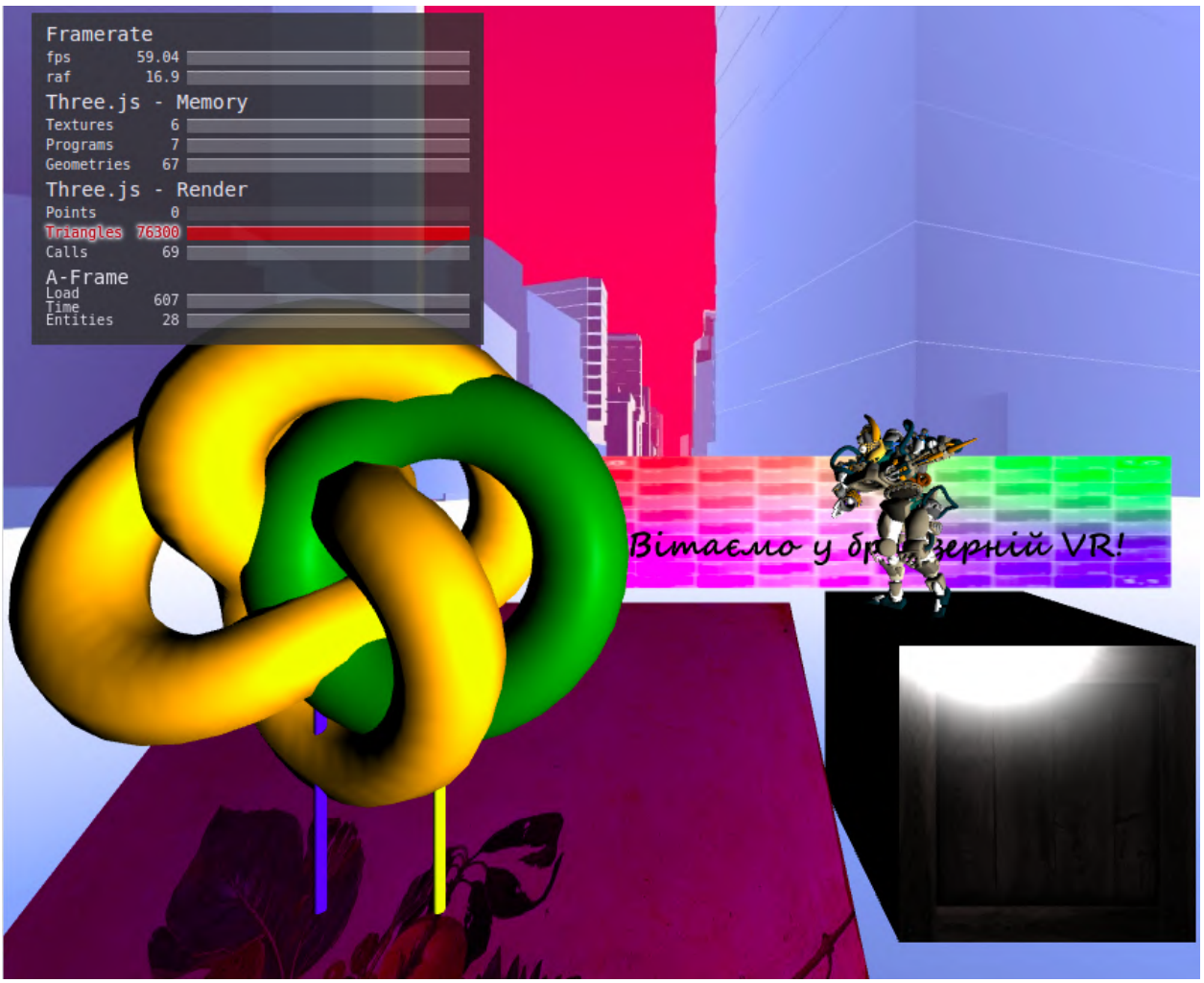

Рис. 3.14. Оновлена сцена на декстопі.

`animation-mixer`, який в найпростішому випадку просто додається до `<a-gltf-model>`.

Параметр `stats`, доданий до сцени, надає можливість відслідковувати споживання ресурсів (рис. 3.14).

3.11 Сцена у віртуальній та доповненій реальності

Випробування побудованої сцени на мобільному пристрої (рис. 3.15) відкриває можливості для переходу до режиму віртуальної та доповненої реальності за кнопками VR (рис. 3.16) та AR відповідно.

За натисканням кнопки AR відбудеться перехід до режиму доповненої реальності, проте за наявності 360° зображення (`<a-sky>`) та відео (`<a-videosphere>`) побачити зображення з камери не вдасться через його перекриття, тому пропонуємо для режиму AR закоментувати відповідні

Рис. 3.15. Оновлена сцена у мобільному браузері.

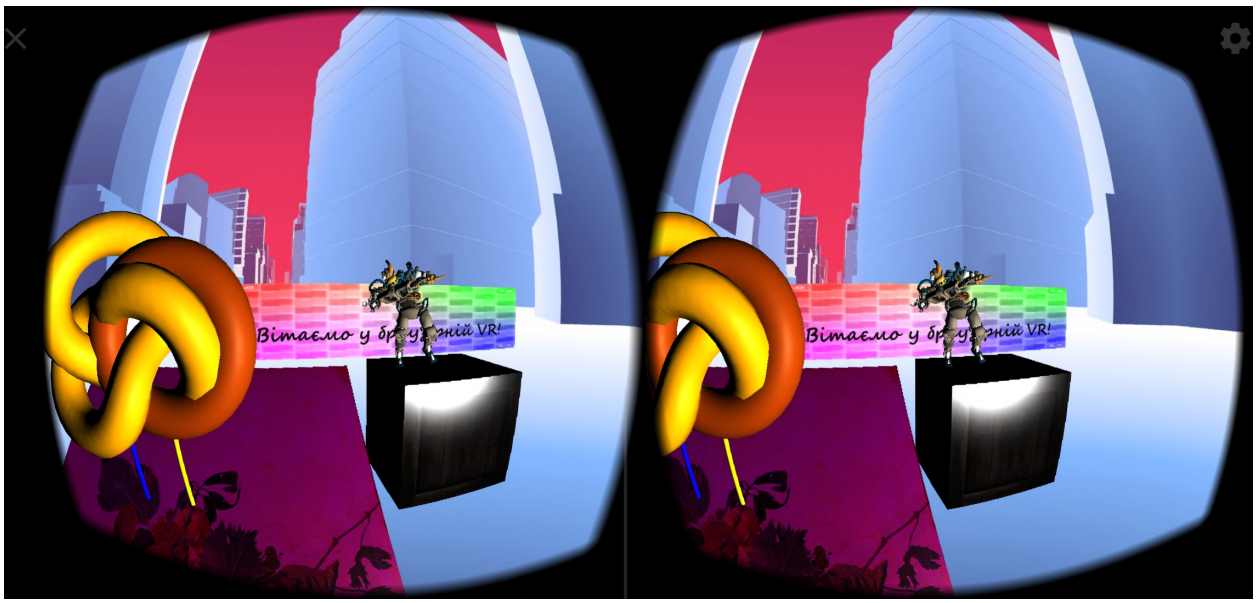

Рис. 3.16. Оновлена сцена в режимі віртуальної реальності.

Рис. 3.17. Оновлена сцена в режимі доповненої реальності.

рядки (рис. 3.17).