КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Курсова робота
за спеціальністю
121 Інженерія програмного забезпечення: програмна інженерія
на тему:

3D графіка та анімація у прикладному застосуванні

Виконала студентка 3-го курсу
Карпишин Ольга Богданівна _____

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Катеринич Лариса Олександрівна _____

(підпис)

Засвідчую, що в цій курсовій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

(підпис)

Київ – 2020

РЕФЕРАТ

Обсяг роботи 48 сторінок, 25 використаних джерел, 21 рисунок, 1 таблиця.

Ключові слова: 3D графіка, анімація, моделювання, рендеринг, фреймворк, браузерна гра, бібліотеки, WebGL.

Об’єктом роботи є процес застосування засобів для візуалізації 3D моделей, опису та реалізації фізичної активності цих моделей. Предметом роботи є браузерна 3D гра, створена мовою JavaScript на базі одних із найбільш популярних бібліотек – Three.js та Cannon.js.

Метою курсової роботи є ознайомлення з концепцією комп’ютерної графіки, особливостями її розвитку і застосування, порівняння підходів та концепцій щодо використання тривимірної графіки у сферах людської діяльності, робота з бібліотеками JavaScript по забезпеченню функціональності 3D об’єктів, а також створення на базі отриманих знань власної браузерної гри.

Інструментом розробки обрано Visual Studio Code – сучасний редактор програмного коду, розроблений компанією Microsoft, зручний для багатоплатформної розробки веб-застосунків. Редактор містить відлагоджувач, інструменти для роботи з Git, підсвічування синтаксису, IntelliSense і засоби для покращення коду (рефакторинг). Мова програмування в середовищі розробки – JavaScript.

Результати роботи: під час виконання курсової роботи було розглянуто та досліджено засоби для моделювання 3D об’єктів, опрацьовано інформацію про можливості графічних бібліотек. У ході написання курсової роботи підготовлено практичну частину: браузерну гру.

Програмний продукт, основу гри у вигляді фреймворку, можна використовувати у майбутньому, розширюючи та доукомплектовуючи функціонал. Гра буде корисною користувачам, оскільки вона добре розвиває моторику рук та швидкість реакції.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

Від англ. – розшифрування абревіатури на англійській мові;

3D (від англ. – three-dimensional) – тривимірний;

HTML (від англ. Hypertext Markup Language) – стандартна мова розмітки для створення веб-сторінок і веб-додатків;

SIGGRAPH (від англ. – Special Interest Group on Graphics and Interactive Techniques) – спеціальна група з графічних та інтерактивних методів; щорічна конференція з питань комп'ютерної графіки, яка проводиться організацією ACM SIGGRAPH

WebGL (від англ. – Web-based Graphics Library) – програмна бібліотека, призначена для створення інтерактивної тривимірної графіки у веб-браузерах.

FPS (від англ. – frames per second) – кількість кадрів на екрані монітора чи телевізора за секунду.

Canvas (від англ. canvas – полотно) – елемент HTML5, який можна застосовувати для малювання графіки, використовуючи скрипти.

ВСТУП

Оцінка сучасного стану об’єкта розробки. Із розвитком високих технологій 3D графіка та анімація набули величезної популярності. Саме вони дали змогу створювати об’ємні та анімовані моделі, починаючи від макетів окремих предметів і продовжуючи проекціями цілих віртуальних світів. Для підтримки належного рівня якості в сучасному технологічному та конкурентному середовищі, відбувається постійне удосконалення даної сфери. Окрема увага приділяється розробці ігор, адже саме вони є одним із основних рушіїв стрімкого прогресу графіки: розвиваються сюжети, персонажі й навколишнє середовище стають усе більш деталізованими, а дії ускладнюються різноманітністю та видовищами.

Серед розроблених програмних продуктів для 3D графіки (програми моделювання): 3D Max, Maya, Blender, Softimage, Houdini, Modo та інші. Для браузерних 3D ігор існують різні плагіни та графічні бібліотеки. Багато застосунків дозволяють розробникам вбудовувати у браузери, що підтримують HTML5, повноцінну 3D графіку безпосередньо, не вдаючись до підтримки плагінів (наприклад, WebGL).

Актуальність роботи та підстави для її виконання. Важливою сферою застосування 3D графіки є браузерні ігри. Вони набули широкого розповсюдження серед користувачів мережі Інтернет, оскільки використовують браузерний інтерфейс і, як правило, не передбачають встановлення додаткового програмного забезпечення. Такі ігри потребують лише наявності браузера та, за необхідності, відповідних плагінів для нього (для прикладу, Flash, Java). Мови програмування, на яких пишуть такі ігри (JavaScript, PHP, Python та інші), добре укомплектовані і пропонують цілий спектр можливостей для їхнього застосування. Вони займають перші місця у рейтингах мов програмування завдяки тому, що мають пряме відношення до 3D графіки й анімації. Саме тому,

спеціаліст, який добре володіє навичками роботи у даному напрямку, високо цінується роботодавцями, інвесторами, чи бізнес-партнерами.

Мета й завдання роботи. Беручи до уваги сучасні тенденції та розвиток засобів для розробки браузерних 3D ігор, метою даної курсової роботи є структуризація знань про комп'ютерну графіку, огляд наявних технологій та вивчення можливостей для їх практичного застосування. Для досягнення мети поставлено такі завдання:

- Ознайомитися із базовою концепцією тривимірної графіки;
- Дослідити наявні засоби для роботи з нею у браузерних іграх;
- Виокремити переваги та недоліки використання програмних продуктів;
- Розробити практичне завдання для продукту;
- Використати набуті знання для створення власної браузерної 3D гри.

Об'єкт, методи й засоби дослідження та розроблення. Об'єктом роботи є процес застосування засобів для візуалізації 3D моделей, опису та реалізації фізичної активності цих моделей.

Розробці практичної частини передувало ознайомлення із концепцією створення браузерних ігор, розробка моделі гри та її структуризація, обґрунтування доцільності використання графічних бібліотек.

Під час розробки програмного продукту враховувалися сучасні тенденції та рекомендації, які можна знайти на веб-сайтах для програмістів. Зокрема, опрацьовано статті, в яких зроблена оцінка щодо популярності того чи іншого застосунку і його функціоналу, а також розглянуто переваги та недоліки цього ресурсу під час практичного застосування.

Інструментом для створення гри було обрано Visual Studio Code від компанії Microsoft, що підтримує мову програмування JavaScript. Він пропонує широкий вибір бібліотек та фреймворків, що стосуються роботи з тривимірною графікою.

Можливі сфери застосування. На базі реалізованої практичної частини курсової роботи можна створювати браузерні 3D ігри, використовуючи можливості вбудованих бібліотек. Програмний продукт, основу гри у вигляді фреймворку, можна використовувати у майбутньому, розширюючи та доукомплектовуючи функціонал. Гра буде корисною користувачам, оскільки вона добре розвиває моторику рук та швидкість реакції.

РОЗДІЛ 1. ЗАГАЛЬНИЙ ОГЛЯД 3D ГРАФІКИ ТА АНІМАЦІЇ

1.1 Історія виникнення комп'ютерної графіки та основні задачі, що вона мала вирішувати

Прийнято вважати, що термін «комп'ютерна графіка» було придумано Вільямом Феттером – американським дизайнером, який, починаючи з 1960-го року, досліджував перспективні основи комп'ютерної анімації у вигляді каркасу фігури людського тіла. Ця тривимірна модель отримала назву «Boeing Man» і стала «пілотом» у короткометражній анімації 1964-го року. Сам Вільям Феттер приписує авторство терміну своєму колезі – Верну Хадсону [1].

У другій половині 1960-х років американські вчені Девід Еванс та Айван Сазерленд започаткували першу кафедру комп'ютерної графіки в Університеті Юти, США, де, за рахунок фінансової допомоги від ARPA та наявності провідних фахівців, було розроблено базові технології 3D графіки, зокрема затемнення по Гуро, Фонгу та Бліну, текстурування, алгоритми виявлення невидимої поверхні. Серед експертів, які працювали на кафедрі, були такі видатні особистості, як Джим Блінн, Буй Тіонг Фонг та Анрі Гуро, які зробили значний вклад у розвиток алгоритмів текстурування і затемнення. Згодом, у 1962 році, Сазерленд створив у лабораторії Массачусетського технологічного університету програму Sketchpad, яка передувала всім сучасним 3D-редакторам та CAD-системам. Для малювання векторних фігур на дисплеї у Sketchpad використовувалося світлове перо, але його визначальною особливістю стало введення концепції об'єктів та екземплярів: їх можна було зберігати, редагувати і застосовувати повторно в інших роботах. Не менш важливим було вміння програми автоматично промальовувати фігури із заданими координатами [2].

У 1968 році Айван Сазерленд разом із Девідом Евансом заснували компанію Evans & Sutherland. Компанія сфокусувала свою діяльність на виробленні новітнього обладнання, що підтримувало роботу систем, які

проектувалися в університеті, де працювали її засновники. Значна частина цих алгоритмів була пізніше імплементована в головному апаратному забезпеченні, зокрема геометричний конвеєр, дисплей з кріпленням до голови, кадровий буфер та авіаційний тренажер. Основу працівників компанії складали випускники, або студенти кафедри, серед яких ми сьогодні можемо зустріти і Джима Кларка, засновника компанії Silicon Graphics, і Едвіна Кетмелла, який став президентом і технічним директором відомої анімаційної студії Pixar, на рахунку якої – перший повнометражний мультфільм, виконаний у редакторах та програмах для тривимірного простору, а також Джона Варнока, одного із основоположників Adobe Systems. [3]

У той самий час, група спеціалістів СРСР на чолі з Миколою Константиновим за допомогою обчислювальної машини БЕСМ-4 змогли змоделювати рухи кішки, використовуючи системи диференціальних рівнянь 2-го порядку. Таким чином, 1968 рік ознаменувався виходом першого мультфільму з анімованим комп'ютерним персонажем «Кішечка»: кадри друкувалися на принтері, а потім об'єднувалися у стрічку [4].

Величезні запити з різних галузей діяльності вимагали все більшого технічного прогресу, тому потрібно було оптимізувати алгоритми для графічних об'єктів і створити зручні інструменти для роботи з ними.

Перш за все, цього потребувала кіноіндустрія. У 1973 році вийшов перший фільм «Дикий Захід», у якому було застосовано цифрову обробку зображень. Після цього уперше використано каркасні тривимірні зображення у продовженні фільму в 1976 році. Авторами картини були Едвін Кетмелл та Фред Парке, які у 1972 році створили експериментальну роботу «Комп'ютерна анімація руки», що стала проривом у сфері 3D графіки. У короткометражці застосовувалися новітні техніки для генерування комп'ютером руки та обличчя – 3D рендеринг, який ми досі використовуємо у відеоіграх, фільмах та спецефектах [5].

Якщо ж говорити про іншу, не менш важливу, галузь – автомобілебудування, то знаменною подією стала розробка П'єром Безьє у 1962

році на замовлення компанії Renault надзвичайно простого і збірного опису будь-яких форм площини, необхідних для автоматизованої роботи механізмів по обробленню листового металу. Його система параметрично заданих кривих залягла у фундамент багатьох графічних програм [6].

На початку 1970-х років виникло питання перенесення на комп'ютер графічних тривимірних зображень, максимально наближених до реальних. За вирішення цієї проблеми взявся послідовник Айвана Сезерленда – Анрі Гуро. У 1971 році він розробив алгоритм, що дозволяв прорисовувати тіні, зберігаючи плавність у переходах між різними кольорами, залежно від віддаленості джерела світла та кута падіння променів. Цей метод створював ілюзії гладкої криволінійної поверхні, що представлена у вигляді полігональної сітки із плоскими гранями, шляхом інтерполяції кольорів сусідній граней. Через два роки його колега, Буй Тіонг Фонг презентував свою техніку тонування об'єктів, що базується на інтерполяції нормалей поверхні та растеризованими полігонами і обраховує колів пікселів за допомогою інтерпольованої нормалі та моделі відбивання світла [7]. Також варто згадати демонстрацію на тематичній конференції SIGGRAPH у 1975 році моделі чайника авторства Мартіна Ньюллема. У попередній рік він опублікував дисертацію «Алгоритм моделювання підрозбиттів при створенні вигнутих поверхонь на екрані комп'ютера», в якій дослідив такі фундаментальні питання, як накладення текстури, B-кубічні фрагменти і Z-буфер. І нарешті, у 1978 році Джеймс Блінн спроектував технологію правдоподібної візуалізації 3D об'єктів – рельєфне текстурування, яка повинна була відтворювати нерівність поверхонь [8].

Алгоритми побудови максимально реалістичного тривимірного зображення постійно удосконалювалися і професіонали з усього світу і сьогодні продовжують цю справу, адже 3D графіка та анімації мають настільки широкий спектр застосування, що уявити сучасний світ без них практично неможливо.

1.2 Створення тривимірного зображення

1.2.1 Вступ у 3D графіку та переваги її використання

Традиційно побудова об’єкта здійснюється у двовимірному просторі на осях X та Y, при цьому видно лише одну сторону зображеного предмета. У тривимірному середовищі зображення представлене ще й третьою віссю – Z – віссю глибини. З її допомогою отримується інформація про всі сторони предметів.

Перша перевага даного методу полягає у тому, що при наявності моделі в 3D художнику достатньо розмістити її в кадрі і анімувати, а фінальне зображення отримується на спеціальній програмі – візуалізаторі.

Друга перевага в тому, що одна тривимірна модель може бути використана повторно в різних проектах, її можна легко змінювати, деформувати і надавати бажаного зовнішнього вигляду. Зі звичайним двовимірним малюнком такі дії не завжди доступні.

Третьою перевагою є можливість проектування деталізованих моделей. На загальному плані це видно не завжди, але варто приблизити камеру і візуалізатор вирахує, що видно у кадрі, а чого немає.

Найбільш популярним способом моделювання є полігональне. Суть даного методу полягає у тому, що поверхня моделей задається за двовимірних геометричних примітивів – різних многокутників. Многокутники, що формують модель, називаються полігонами. Як правило, під час розробки 3D-моделей, надають перевагу використанню чотирикутників через те, що при експорті моделей в ігровий рушій їх досить легко перетворити у трикутники, а за необхідності згладжування, модель з чотирикутників формується без візуальних артефактів. Чим більше полігонів містять модель, тим більше ця модель схожа на оригінал. Однак, у значній кількості полігонів є мінус – зниження продуктивності. Велике число полігонів відповідає великому числу точок, за

якими їх будують, що призводить до збільшення кількості даних, які має обробити процесор. Тому під час створення моделей часто доводиться йти на компроміс між продуктивною та деталізованою моделями. У зв'язку з цим виникли такі терміни, як високополігональна модель і низькополігональна модель. В комп'ютерних іграх застосовують низькополігональні моделі, оскільки візуалізація в іграх виконується в реальному часі [9].

Після етапу моделювання отримуємо математичну модель, яка містить лише інформацію про геометричну форму об'єкта. Для того, щоб модель була потрібного кольору і мала здатність віддзеркалювати, використовують текстури.

Текстура – це двовимірний малюнок, який накладається на тривимірну модель. Текстури бувають процедурними (тобто, згенерованими за допомогою алгоритму) і намальованими в графічному редакторі. Текстура задає тільки малюнок і колір моделі, а здатність віддзеркалення, переломлення, рельєф і прозорість задаються у властивостях матеріалу. З точки зору тривимірної графіки, матеріал – це математична модель, яка описує параметри поверхні. Перед тим, як накласти текстуру на модель, необхідно зробити її розгортку, тобто, уявити поверхню моделі у вигляді проекції на площину. Таким чином, створення тривимірної моделі складається з таких стадій:

- Відмальовка ескізу моделі, або ж пошук зображення того, з чого буде створена модель;
- Моделювання геометричної форми об'єкта на основі ескізу або зображення;
- Створення розгортки;
- Створення текстур;
- Налаштування параметрів матеріалу (текстур, відображення, заломлення, прозорості і т.д.) [10].

Після виконання цих пунктів модель готова для візуалізації, або ж із нею можна продовжувати роботу з налаштування ріггінга (rigging) моделі, чи створення анімації.

Після виготовлення моделі в 3D графіці, її потрібно помістити в сцену до інших об’єктів, додати камеру, освітлення, і тоді можна отримати фінальне зображення. Це зображення прораховується на основі фізичної моделі – моделі поширення променя світла з урахуванням відображення, заломлення, розсіювання. При традиційному 2D малюванні художник самотужки зображує відблиски, тіні тощо. У 3D графіці автор створює сцену з урахуванням геометрії, матеріалів освітлення, властивостей камери, а візуалізатор вираховує кінцеве зображення.

1.2.2 Аналіз сучасних методів 3D моделювання

3D системи працюють з трьома координатами так, що зміна одного виду призводить до змін на інших. Певні системи мають у своєму розпорядженні технології аналізування, що автоматично вираховують такі характеристики фізичних властивостей тіла, як момент інерції, чи вага.

Виділяють три види моделювання у тривимірному просторі [10]:

- Каркасне;
- Твердотільне;
- Поверхневе.

Каркасна модель цілковито описується термінами ліній і точок. Цей спосіб моделювання має великий ряд обмежень і вважається методом моделювання найнижчого рівня. Більшість обмежень з’являться внаслідок недостачі даних про грані, які розташовані між лініями, і неможливості виділити окремо внутрішню та зовнішню частини зображення об’ємного твердого предмета.

Серед недоліків цієї моделі можна виділити:

- Сумнівність результатів – потреба у заданні всіх ребер для коректного відображення моделі;
- Кількісно малий клас об’єктів для моделювання через виключення можливості розпізнавати криволінійні грані;

- Відсутність інформації про взаємне розташування двох об’єктів;
- Неточності при розрахунку фізичних властивостей тіла;
- Неможливість затонування зображень через відсутність інформації про грані.

З іншого боку, говорячи про переваги, каркасне моделювання потребує менших затрат ресурсів комп’ютера та підходить для вирішення нескладних завдань. Часто цей метод використовується не в моделюванні, а при відображенні готових моделей, як один із методів візуалізації (Рис.1.2.1).

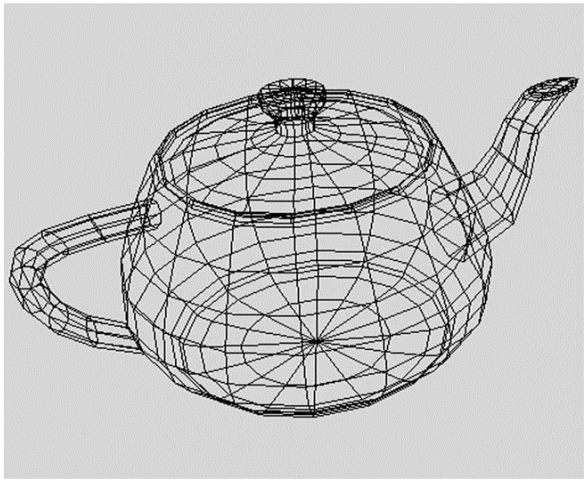


Рис. 1.2.1 Каркасна модель чайника

Наступним методом моделювання є поверхневе моделювання (Рис. 1.2.2), яке визначається в термінах точок, ліній і поверхонь. Метод ґрунтується на тому, що об’єкти обмежуються поверхнями, які відокремлюють їх від довкілля. Ця оболонка зображена графічними поверхнями. Поверхня об’єкта обмежена контурами, які є результатом двох дотичних або поверхонь, що перетинаються. Вершини об’єктів задаються перетином трьох поверхонь.

При виборі поверхневого моделювання є певні плюси, порівнюючи із каркасним:

- Можливість визначення складних криволінійних граней;
- Здатність отримання тонових зображень;

- Розпізнавання особливих побудов на поверхні, наприклад, отворів;
- Отримання зображення високої якості.

В основі моделювання поверхні лежать два математичних визначення: усяка поверхня може апроксимуватися многогранником, кожна грань якого складена з найпростішого плоского многокутника. Також в моделі припустимі поверхні другого порядку і поверхні, форма яких визначається завдяки різним способам апроксимації і інтерполяції. Будь-який об’єкт, сформований за допомогою поверхневого моделювання, має внутрішню і зовнішню сторону [10].

Є декілька типів поверхонь [11]:

- Базові геометричні поверхні – це поверхні, які виходять при переміщенні однієї плоскої кривої щодо іншої;
- Поверхні обертання – це поверхні, створені при обертанні плоскої грані навколо деякої осі;
- Поверхні об’єднання і перетинів – це поверхні, створені внаслідок об’єднання або перетину поверхонь;
- Скульптурні поверхні – поверхні, які ніяк не описуються математичними рівняннями і будуються лише при використанні сплайнів, що сполучають точки в просторі;
- Аналітичні поверхні – поверхні, описані математичним рівнянням.

Рис 1.2.2 Поверхневе моделювання

Останній метод моделювання – твердотільне моделювання. Твердотільна модель (Рис. 1.2.3) визначається термінами того тривимірного об’єму, який займає тіло, що її визначає. Твердотільне моделювання – найдосконаліший і найбільш достовірний метод створення реального об'єкта. У цього методу є ряд переваг:

- Можливість розділення внутрішньої і зовнішньої областей об’єкта;
- Приховані лінії видаляються автоматично;
- Автоматичне створення тривимірних розрізів об'єкта, що важливо при аналізуванні складних виробів;
- Використання методів аналізу з автоматичним отриманням зображення конкретних властивостей;
- Здатність отримання тонових зображень;

Є два способи створення твердотільних моделей [10]:

- Метод конструктивного представлення;
- Метод граничного представлення.

Метод конструктивного представлення зводиться до того, що з базових складових елементів (твердотільних примітивів, які визначаються формою, розмірами, точкою прив’язки і орієнтацією) створюється твердотільна модель. Твердотільні моделі створюються за допомогою булевих операцій: об’єднання, перетин, різниця. У даного методу є свої переваги і недоліки. За допомогою методу конструктивного представлення можна забезпечити хорошу точність розрахунків фізичних властивостей об’єкта, однак цей метод відмінний від звичних способів моделювання і є складним у застосуванні без відповідного досвіду роботи.

Другий метод, метод граничного представлення, робить опис границь об’єкта і чітко задає межі, що описують тіло. Лише цей метод дозволяє створити точне представлення геометричного твердого тіла. При цьому підході потрібно задати контури або межі об’єкта і ескізи різних видів на об’єкт, а також вказати лінії зв’язків між цими видами для установки взаємовідповідності.

Порівнюючи обидва методи можна сказати, що метод конструктивного представлення більш зручний при первісній побудові моделі та займає менше місця в базі даних. Однак він, на відміну від методу граничного подання, не дуже зручний для побудови складних форм. Також, хоча в методі конструктивного представлення дані займають менше місця, обсяг розрахунків при відтворенні моделі виявляється більшим. З іншого боку, метод граничного представлення зберігає точний опис границь моделі, який займає багато пам’яті, але майже не вимагає обчислень для відтворенні моделі. Перевагою методу граничного представлення також є простота перетворення у каркасну модель і в зворотному напрямку.

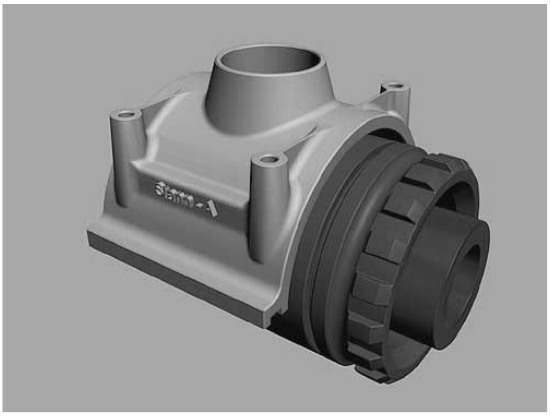


Рис 1.2.3 Твердотільне моделювання

Поєднання конструктивного і граничного представлень є гібридною системою. Гібридне моделювання дозволяє поєднувати каркасну, твердотільну і поверхневу геометрії, а також використовувати параметричне моделювання.

РОЗДІЛ 2. ЗАСОБИ ДЛЯ ВПРОВАДЖЕННЯ 3D ГРАФІКИ У ВЕБ-СЕРЕДОВИЩЕ

2.1 Особливості розробки під WebGL

Як пишуть розробники Khronos Group [13], WebGL – це багатоплатформний, безкоштовний API для підтримки 3D графіки в браузері, що використовує мову програмування шейдерів GLSL. WebGL дозволяє створювати 3D компоненти через Canvas і виконується як елемент HTML5, тому є повноцінною частиною об'єктної моделі документа (DOM API) браузера (Рис. 2.1.1). Може застосовуватися з будь-якими мовами програмування, що підтримують роботу з DOM API – JavaScript, Java, Kotlin, Rust. Автоматичне управління пам'яттю надається мовою JavaScript.

На даним момент існують дві його версії:

- WebGL 1.0 – будується на базі OpenGL ES 2.0, для шейдерів підтримується мова GLSL ES версії 1.00
- WebGL 2.0 – будується на базі OpenGL ES 3.0, для шейдерів підтримується мова GLSL ES версії 1.00 і 3.00

Підтримку WebGL реалізовано в таких браузерах:

- Mozilla Firefox: WebGL вбудований у всіх платформах, які мають необхідну графічну карту з актуальними драйверами, починаючи з версії 4.0;
- Google Chrome: WebGL вбудований по замовчуванню у всіх версіях, починаючи з дев'ятої;
- Safari: експериментально підтримує WebGL, починаючи з версії 5.1, повна підтримка реалізовано і вбудовано за замовчуванням у версії 8.0;
- Opera: WebGL реалізований у версії Opera 12.0, але не вбудований за замовчуванням.

- Internet Explorer: починаючи з Internet Explorer 11, WebGL офіційно підтримується. До виходу 11 версії незалежними розробниками були випущені плагіни Chrome Frame і IEWebGL, що передбачають опції, необхідні для підтримки WebGL в Internet Explorer.

Серед головних переваг WebGL можна виділити:

- Багатоплатформенність – розроблену програму зможуть побачити всі користувачі різних платформ за допомогою браузера;
- Розробка на JavaScript;
- Можливість очищення пам’яті від «сміття», що оптимізує роботу;
- WebGL має вшитий процесор та відеокарту для оброблення графічних програм.

Рис. 2.1.1 Приклади роботи з WebGl

2.2 Популярні бібліотеки, рушії та фреймворки JavaScript для роботи з 3D графікою

2.2.1 Огляд бібліотек

Сучасні браузери мають багато суттєвих покращень у порівнянні зі своїми попередниками. Зараз вони можуть показувати інтерактивні 3D сцени зі складними об’єктами і фотореалістичною візуалізацією. Є чимало бібліотек, заснованих на WebGL і CSS. Нижче наведено короткий огляд деяких із них.

Voxel

Легка, заснована на CSS 3D бібліотека з простим набором класів. У бібліотеці 4 основні класи: сцена, навколишній світ, редактор і безпосередньо воксель – елемент простору, що позначає значення певної величини у клітинках рівномірної просторової гратки. Клас «сцена» відповідає за розташування камери, «навколишній світ» управляє розміщенням вокселей, що додаються до нього, а « редактор» дозволяє користувачеві маніпулювати положенням камери і виконувати дії над вокселем [14].

Photon

JavaScript бібліотека, що додає 3D ефекти до різних об’єктів. Досить сильно навантажує процесор, про що слід пам’ятати, якщо необхідно, щоб сайт адекватно працював на малопотужних комп’ютерах [15].

Sprite3D.js

Бібліотека дозволяє маніпулювати HTML елементами в тривимірному просторі. Можна керувати розташуванням, обертанням і масштабуванням елементів за допомогою простих функцій, які можна застосовувати послідовно (у вигляді конвеєра). Об’єкти Sprite3D є звичайними HTML елементами, тому до них застосовуються стандартні CSS директиви [16].

У наступному підрозділі детально описано одну з найперших, найпоширеніших та найбільш укомплектованих відкритих бібліотек – Three.js, яку використано у практичній частині даної курсової роботи. Її вибрано тому, що вона пропонує якісну технічну документацію, багато статей, уроків та прикладів для її вивчення та використання.

2.2.2 Функціонал бібліотеки Three.js

Рис. 2.2.2.1 Приклад роботи Three.js (geometry/text) [17]

Як показує практика, процес роботи із WebGL є доволі громіздким, особливо коли йдеться про шейдери. Також, під час розробки моделі необхідно описати всі складові: точки, лінії, грані тощо.

Теоретично, для візуалізації ідеї потрібно було б написати великий код, що не є ефективно з точки зору часу і продуктивності. Саме тому, для прискорення цього процесу розробили JavaScript бібліотеку – Three.js. [18]. Головним автором бібліотеки вважають Рікардо Кабелло, але участь у її розробці взяли багато професіоналів.

Бібліотека містить цілий ряд готових класів для реалізації та відображення тривимірної графіки у WebGL [24]. Засоби Three.js дозволяють використовувати звичні терміни, не удаючись до написання шейдерів.

Головні поняття, якими оперує бібліотека [25] (Рис. 2.2.2.2):

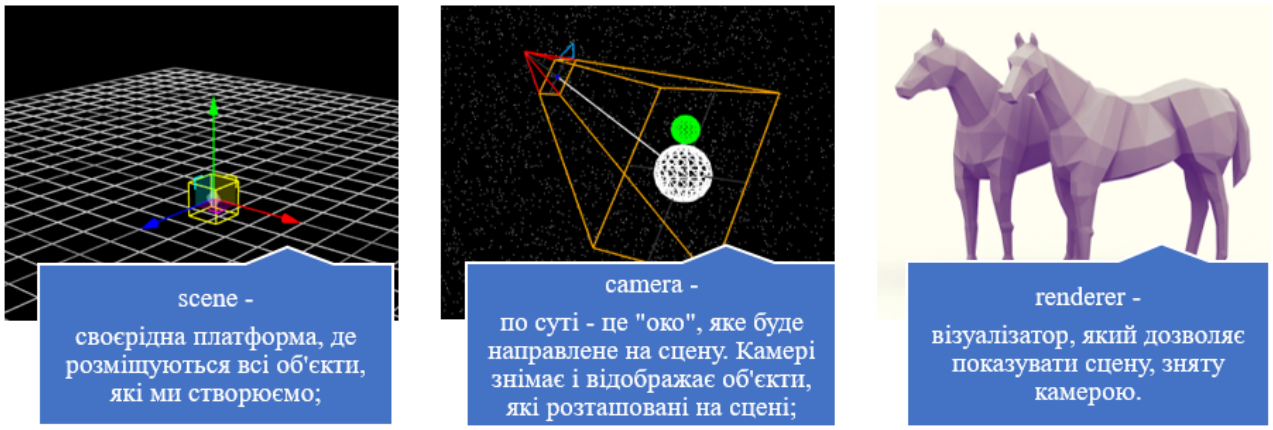


Рис. 2.2.2.2 – Базові поняття бібліотеки

У Three.js існує кілька типів камери (найчастіше застосовують перший ряд):

- Perspective Camera
- Orthographic Camera
- Stereo Camera
- Cube Camera

Perspective Camera (Рис. 2.2.2.3) – найбільш поширений режим проекції, який використовується для візуалізації 3D-сцени. Перспективна камера призначена для імітації того, що бачить людське око. Вона сприймає об’єкти в перспективній проекції, тобто, чим дальше знаходиться об’єкт, тим меншим він здається. Камера приймає чотири аргументи:

- *FOV* або *Field Of View* (поле/кут зору) – визначає кут, який можна бачити навколо центру камери.
- *Aspect ratio* – співвідношення ширини до висоти екрану. При великих значеннях поля зору видимий розмір об’єктів швидко зменшується при віддаленні. При малих значеннях, навпаки, видимий розмір об’єктів слабо залежить від відстані.
- *Near & Far* – мінімальна і максимальна відстань від камери, яка потрапляє в рендеринг. Так, дуже далекі точки не будуть вимальовуватися взагалі, як і точки, які знаходяться надто близько.

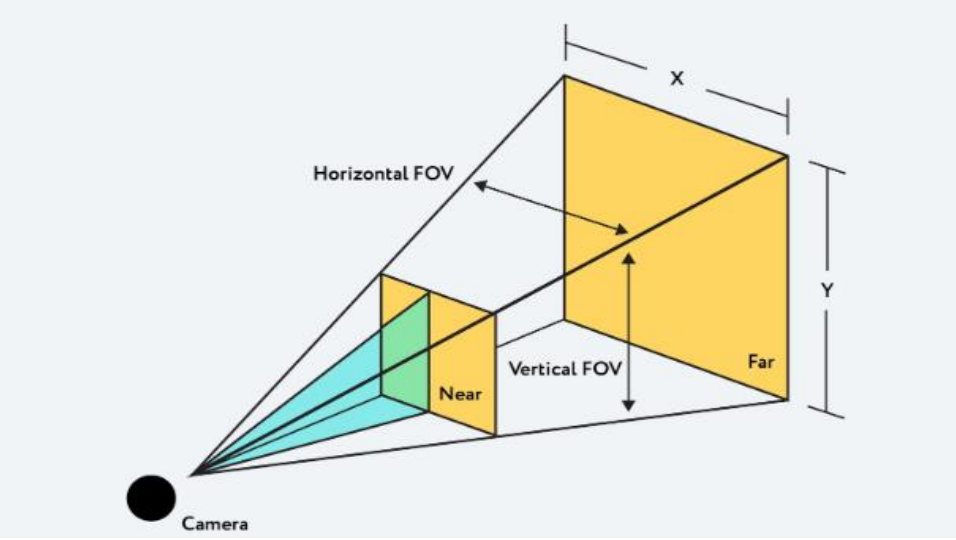


Рис. 2.2.2.3 Ілюстрація роботи перспективної камери

В режимі проектування Orthographic Camera **(**Рис. 2.2.2.4 (б)) розмір об’єкта відображеному зображенні залишається постійним, незалежно від його відстані від камери. Тобто, це камера, віддалена на нескінченну відстань від об’єктів. Усі перпендикулярні прямі залишаються перпендикулярними, а всі паралельні – паралельними. Якщо рухати камеру, прямі і об’єкти не будуть спотворюватися. Це може бути корисним при відображенні 2D сцен і елементів UI

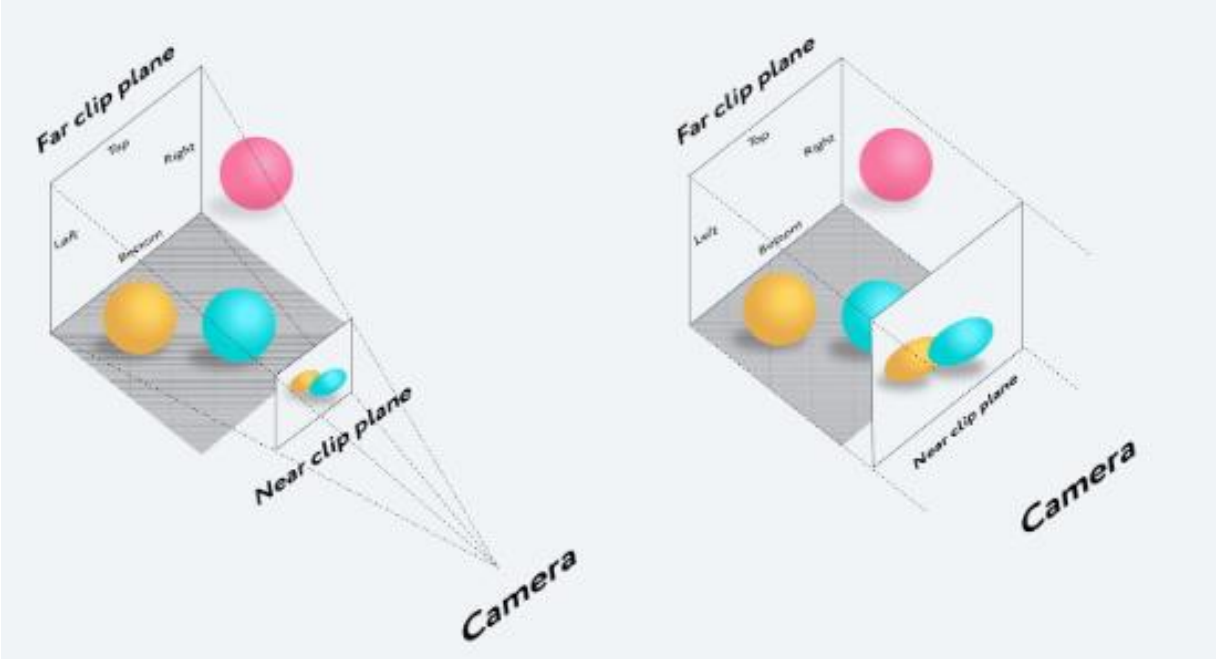


Рис. 2.2.2.4 Перспективна (а) і ортографічна (б) камери

Важливим інструментом є освітлення, без якого на сцені, буде складатися враження, що глядач (користувач) знаходиться в темній кімнаті. Крім цього, за допомогою освітлення можна надати більшу реалістичність сцені. Технічно кожному освітленню можна задати колір. Приклади освітлення (Рис. 2.2.2.5):

- *Ambient Light* – фонове освітлення, яке використовується для освітлення всіх об’єктів сцени однаково; не може бути використано для створення тіней, оскільки не має напрямку.

- *Directional Light* – світло, що випромінюється в певному напрямку. Це світло буде вести себе так, ніби воно нескінченно далеко. Воно може відкидати тіні, оскільки направлене на конкретний об’єкт.
- *Point Light* – світло, що випромінюється з однієї точки в усіх напрямках. Звичайний випадок використання такого освітлення це повторення освітлення від простої лампочки.
- *Spot Light* – світло, яке випромінюється з однієї точки в одному напрямку вздовж конуса, що розширюється по мірі віддалення від джерела світла.

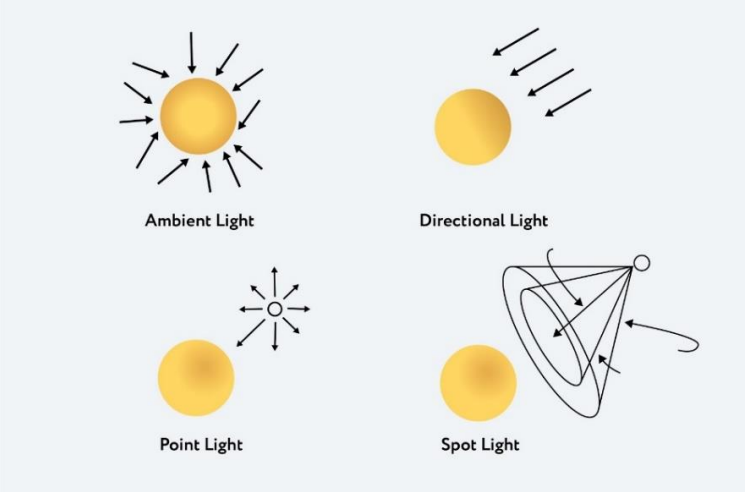


Рис. 2.2.2.5 Приклади освітлень

Тепер перейдемо до питання створення об’єктів на сцені. Об’єкт, що створюється на сцені, називається Mesh. Цей клас приймає 2 аргументи:

- *Geometry* – описує форму (положення вершин, межі, радіус і т.д). (Рис.2.2.2.6)
- *Material* – описує зовнішній вигляд об’єктів (колір, текстура, прозорість і т.д.) (Рис.2.2.2.7).

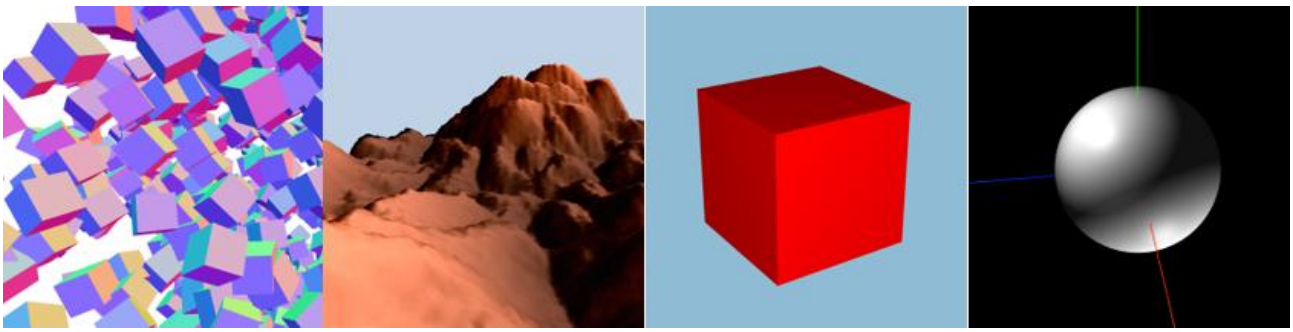


Рис. 2.2.2.6 – Geometry, приклади із офіційного сайту Three.js [17]

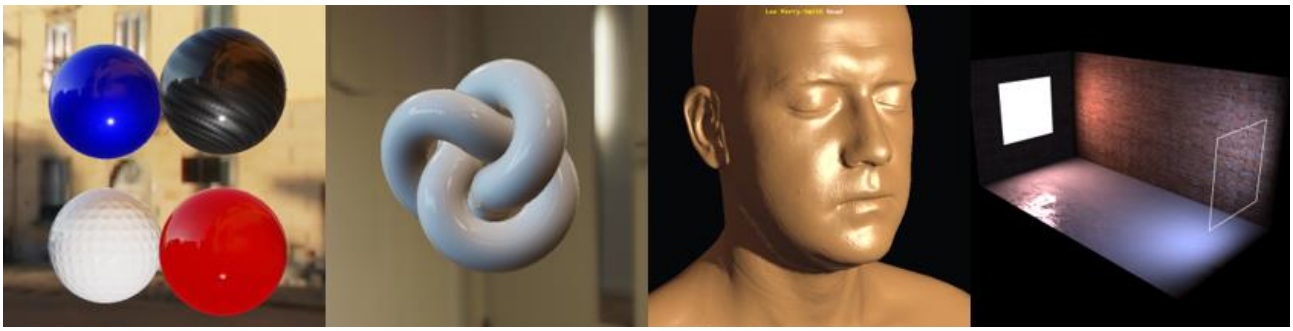


Рис. 2.2.2.7 – Material, приклади із офіційного сайту Three.js [17]

Для наочності розглянемо процес створення засобами Three.js найпростіший фігур – куба (Рис. 2.2.2.7) та сфери (Рис. 2.2.2.8)

Насамперед, переходимо на офіційний сайт бібліотеки https://threejs.org/, завантажуємо її останню версію. Потім підключаємо бібліотеку в секції *head* або на початку секції *body* нашого документа:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=utf-8>
    <title>First Three.js app</title>
    <style>
      body { margin: 0; }
      canvas { width: 100%; height: 100% }
      </style>
  </head>
  <body>
    <script src="js/three.js"></script>
    <script>
      // Тут увесь Javascript код.
    </script>
  </body>
</html>
```

Після цього, аби відобразити створюваний об’єкт, необхідно створити сцену, додати камеру і налаштувати рендер. Додаємо сцену:

```
var scene = new THREE.Scene();
```

Додаємо перспективну камеру:

```
var camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight, 0.1, 1000 );
```

Камера приймає на себе чотири параметри, про які було згадано раніше:

- кут зору, або *FOV*. У нашому випадку це стандартний кут 75;
- співвідношення сторін, або *aspect ratio*;
- третім і четвертим параметром будуть мінімальна і максимальна відстань від камери, яка потрапить в рендеринг.

Додаємо і налаштовуємо рендер:

```
var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );
```

Отже, підсумуємо зроблене: спочатку ми створили об’єкт рендера, потім встановили його розмір відповідно до розміру видимої області і, зрештою, додали його на сторінку, щоб створити порожній елемент *canvas*, з яким будемо працювати. Після створення рендера вказуємо, де потрібно відобразити тег *canvas* . У нашому випадку ми додали його в тег *body*.

Для створення самого куба спочатку задаємо геометрію:

```
var geometry = new THREE.BoxGeometry( 10, 10, 10);
```

Куб створюється за допомогою класу *BoxGeometry*. Це клас, який містить у собі вершини і межі куба. Передаємо розміри:

- *width*: ширина куба, розмір сторін по осі X
- *height*: висота куба, тобто розмір сторін по осі Y

- *depth*: глибина куба, тобто розмір сторін по осі Z

Щоб розфарбувати куб, задаємо матеріал:

```
var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
```

У нашому випадку заданий *MeshBasicMaterial* і переданий параметр кольору *0x00ff00*, тобто, зелений колір. Цей матеріал використовується для надання фігурі однорідного кольору. Недолік його в тому, що у фігури зникає глибина. Але цей матеріал може стати в нагоді при відображенні каркасів за допомогою параметра *{wireframe: true}*.

Тепер нам потрібен об'єкт *Mesh* , який приймає геометрію, і застосовує до нього матеріал:

```
var cube = new THREE.Mesh( geometry, material );
scene.add( cube );

camera.position.z = 25;
```

Додаємо *Mesh* на сцену і віддаляємо камеру, оскільки всі об'єкти після методу *scene.add()* за замовчуванням додаються з координатами *(0,0,0)*, через що камера і куб будуть в одній точці.

Для того щоб анімувати куб, потрібно відмалювати все всередині циклу рендеринга, використовуючи *requestAnimationFrame*:

```
function render() {
    requestAnimationFrame( render );
cube.rotation.x += 0.01;
cube.rotation.y += 0.01;
renderer.render( scene, camera );
}
render();
```

requestAnimationFrame – це запит до браузеру про те, що ви хочете щось анімувати. Ми передаємо йому функцію для виклику, тобто функцію *render()*.

Тут же задаємо параметри швидкості обертання. В результаті, цикл рендерить нашу сцену 60 раз в секунду і змушує куб обертатися.

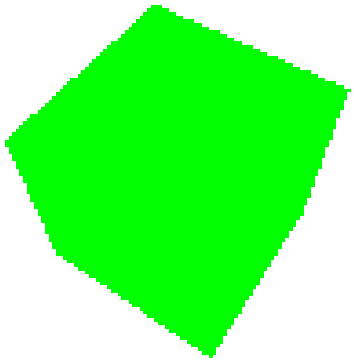


Рис. 2.2.2.7 Куб

Тепер намалюємо сферу.

```
var geometry = new THREE.SphereGeometry(1, 32, 32);
```

Для побудови сфери використовується клас *SphereGeometry*, який приймає на себе:

- радіус (значення за замовчуванням дорівнює 1)
- *widthSegments* – кількість горизонтальних сегментів (трикутників). Мінімальним значенням є 3, значення за замовчуванням 8
- *heightSegments* – кількість вертикальних сегментів. Мінімальним значенням є 2, значення за замовчуванням 6

Чим більша кількість трикутників, тим гладкішою буде поверхня сфери.

Далі спробуємо використовувати інший матеріал – *MeshNormalMaterial* – різнобарвний матеріал, який зіставляє вектори нормалей в *RGB* кольори:

```
var material = new THREE.MeshNormalMaterial();
var sphere = new THREE.Mesh( geometry, material );
scene.add( sphere );
camera.position.z = 3;
```

Видів цього матеріалу існує дуже багато. Деякі з них можна поєднувати і застосовувати одночасно до однієї фігури.

Останнім кроком задаємо цикл рендеринга:

```
function render() {
    requestAnimationFrame( render );
    sphere.rotation.x += 0.01;
  sphere.rotation.y += 0.01;
  renderer.render( scene, camera );
}
render();
```

І отримуємо сферу:

Рис. 2.2.2.8 Сфера

2.2.3 Рушії для створення застосунків із симуляцією фізики та їх порівняння

Ми обрали три рушії для огляду та порівняння їхньої продуктивності [19], [20], [21].

Ammo.js

Порт Bullet physics engine на Javascript з використанням компілятора Emscripten. Функціонал Ammo.js великий, а для роботи з ним знадобиться окрема бібліотека для візуалізації. Найчастіше використовується Three.js, при цьому на кожен цикл перемальовування доведеться вручну синхронізувати положення і обертання кожного об’єкта на сцені з його фізичною моделлю, оскільки рушій не робить це автоматично.

Продуктивність рушія невисока, але помітних осідань fps в більшості проектів не очікується. З недоліків – API, який може бути досить заплутаним і

важко зрозумілим навіть при наявності технічної документації. Однак, в цілому, це хороший інструмент, який продовжує розвиватися і доопрацьовуватися.

Cannon.js

Cannon.js – легкий фізичний рушій з відкритим вихідним кодом. На відміну від попереднього, він від самого початку написаний на Javascript і дозволяє використовувати всі її можливості та оптимізації. Cannon.js в порівнянні з Ammo.js вважається більш компактним, продуктивнішим, а також більш легким для розуміння, але при цьому він не володіє такою великою кількістю функцій. Для роботи також потрібна стороння графічна бібліотека. На практиці продуктивність обох рушіїв приблизно однакова

Oimo.js

Oimo.js – переписана на Javascript версія рушія OimoPhysics. У порівнянні з іншими варіантами, має дуже хорошу продуктивність і точність, проте підтримує тільки примітивну геометрію (куби і сфери). Рушій включений до складу Babylon.js – фреймворка для візуалізації 2D і 3D графіки, тому додаткових бібліотек не потребує. Однак, його мінусом є не дуже якісна документація, але розробники продовжують роботу над нею.

Порівняємо продуктивність рушіїв і подивимося, як вони справляються з обробкою колізій великої кількості об’єктів. Використовуваний браузер – Firefox 64.0.2 x64.

Рушій	**fps при обробці 100 об’єктів**	**fps при обробці 500 об’єктів**	**fps при обробці ніж 1000 об’єктів**
Ammo.js	40-50	25-27	15-25
Cannon.js	30-40	20-25	15-20
Oimo.js	45-55	35-40	35-40

Таблиця 2.2.3.1 Порівняння рушіїв

За результатами тестів (Таблиця 2.2.3.1) кращу продуктивність показує Oimo.js, однак не доцільно робити висновки, відсилаючись лише на такий тест, адже продуктивність також залежить від багатьох сторонніх чинників.

В цілому, вибір конкретного рушія залежить від поставленого завдання. Якщо потрібно простий в розумінні рушій, добре підходять Cannon.js або Oimo.js, тому для практичної частини ми обрали Cannon.js.

2.2.4 Фреймворки A-Frame, Babylon.js

Потужним інструментом для розробки браузерних ігор, є фреймворки. Серед найбільш популярних можна відзначити A-Frame (Рис. 2.2.4.1) і Babylon.js (Рис. 2.2.4.2). Розглянемо їх детальніше.

A-Frame – це веб-фреймворк для створення віртуальної реальності (VR), простий та ефективний спосіб розробки VR-контенту. Він базується на HTML і має потужний набір компонентів, що забезпечують унікальну декларативну структуру з можливістю розширення і компонування. Спочатку інструмент був задуманий в Mozilla, але зараз підтримується на Supermedium. Будучи незалежним проектом з відкритим вихідним кодом, A-Frame перетворився в один з найбільших елементів у всій системі VR.

A-Frame не написаний на чистому WebGL, в його основі лежить бібліотека Three.js. Серед його переваг: полегшена VR розробка, декларативний HTML, компонентно-сутнісна (ECS) архітектура, багатоплатформна VR, продуктивність (A-Frame оптимізована з нуля до WebGL), візуальний інспектор.

Серед основних компонентів A-Frame: геометрія, матеріали, джерела і промені світла, анімація, моделі, тіні, позиційне аудіо, текст і інші елементи управління. A-Frame використовується багатьма провідними компанія світу [22].

Рис. 2.2.4.1 Логотип A-Frame

Babylon.js – легкий кросбраузерний Javascript-фреймворк, який використовує API WebGL для відображення 2D і 3D-графіки в браузері без використання будь-яких сторонніх плагінів і доповнень. Він використовує елемент HTML5 Canvas. Фреймворк поширюється під ліцензією Apache 2. Вихідний код розташований на GitHub. Фреймворк був розроблений Девідом Катуша, Девідом Руссе, і Мішелем Руссо, а також незалежними сторонніми розробниками [23].

Серед його можливостей:

- Сцена: використання готових мешів, туман, скайбоксів;
- Фізичний рушій (модуль oimo.js);
- Згладжування;
- Анімаційний та звуковий рушії;
- Система частинок;
- Апаратне масштабування;
- Підтримка LOD;
- Покрокове завантаження сцени та її автоматична оптимізація;
- Панель налагодження;
- 4 джерела освітлення: точкове, той, що випромінюється всюди, прожектор і реалістичне;

- Користувальницькі матеріали і шейдери;
- Широкі можливості текстурування;
- SSAO;
- Відблиски;
- 9 видів камери, в тому числі і для сенсорного управління;
- Експортери для 3ds Max, Blender, Unity3D, Cheetah 3d;
- Карта висот.

Рис. 2.2.4.2 – Логотип Babylon.js

РОЗДІЛ 3. ТЕХНОЛОГІЇ РОЗРОБКИ ТА СТРУКТУРА ПРОЕКТУ

3.1 Основна мета проекту та підбір технологій

Приступаючи до виконання курсового проекту, ми поставили за мету освоїти базові навички роботи із тривимірною графікою, зокрема створення сцени, налаштування камери, проектування чи імпорт моделей, освітлення, тонування, рендеринг, правильний опис фізики тіл, розробка зручного користувацького інтерфейсу, звуковий супровід для певних дій та в цілому доповнити свої знання про написання коду для браузерних ігор.

Досліджуючи дану сферу, було опрацьовано статті, дописи на тематичних платформах, книги, відео, інструкції щодо вибору засобів для розробки та ін. Окрему увагу було звернено на Three.js – повнофункціональну бібліотеку для полегшення роботи із WebGL, яка стала основою для візуалізації задуманих в

даній роботі ідей, зокрема, завдяки наявності докладно розписаної документації та допоміжних матеріалів для її вивчення у вигляді відеоуроків та інструкцій. Бібліотека відповідає практично за всю візуальну частину проекту (Рис. 3.1.1):

Рис. 3.1.1 – Збірка сегментів, за які відповідає Three.js

Однак, використовувати Three.js для імплементації фізики досить складно, тому ми обрали Cannon.js (Рис. 3.1.2) для таких аспектів гри, як поведінка та взаємодія об’єктів.

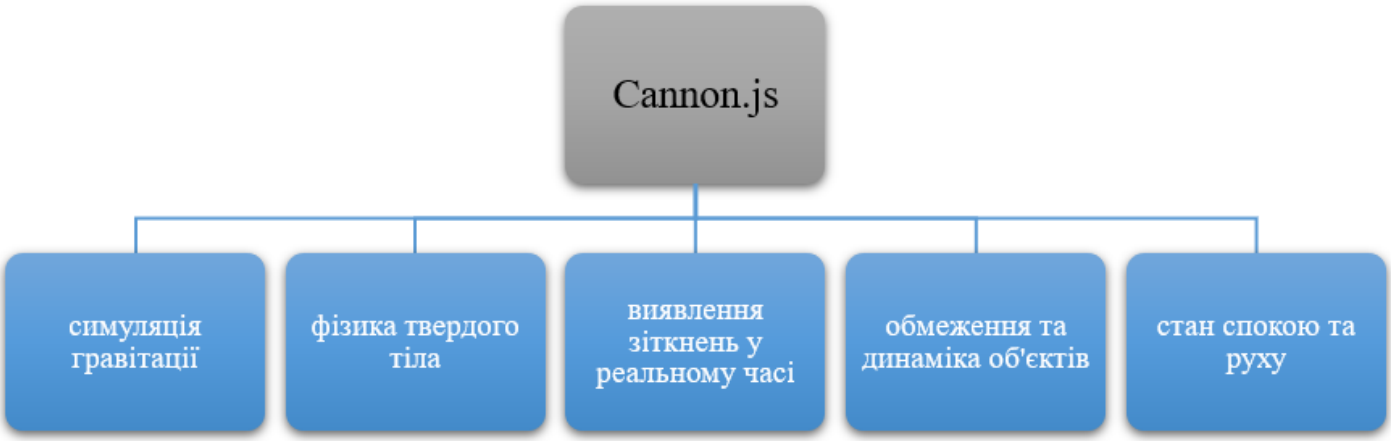


Рис. 3.1.2 – Збірка сегментів, за які відповідає Cannon.js

3.2 Опис створеної браузерної гри

Основу гри складає набір JS файлів, що формують початкову точку зору гри використовуючи систему спостереження камери від третьої особи (Рис. 3.2.1), її перевагою порівнюючи з перспективною є те, що вона переслідує головного гравця:

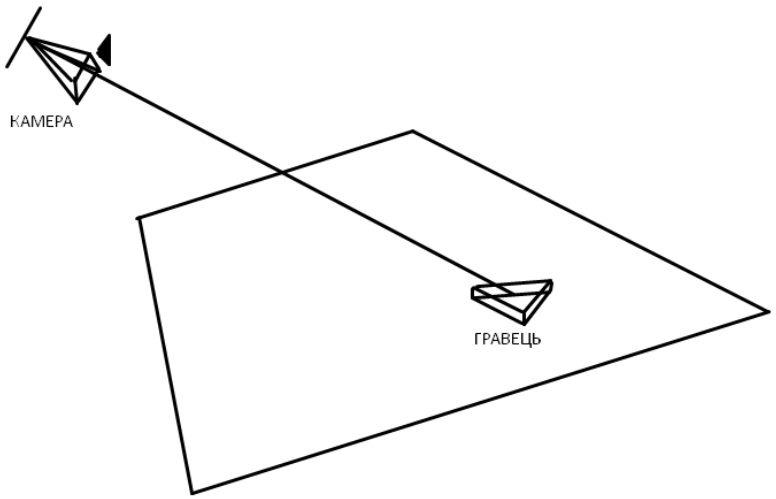


Рис.3.2.1 – Камера від третьої особи

Таку концепцію можна втілити засобами Three.js, адже камері лише потрібно відновлювати попереднє місцезнаходження сітки гравця. Однак, варто забезпечити зв’язок із бібліотекою, що буде контролювати увесь рух об’єктів.

Гру можна розділити на такі складові (Рис. 3.2.2):

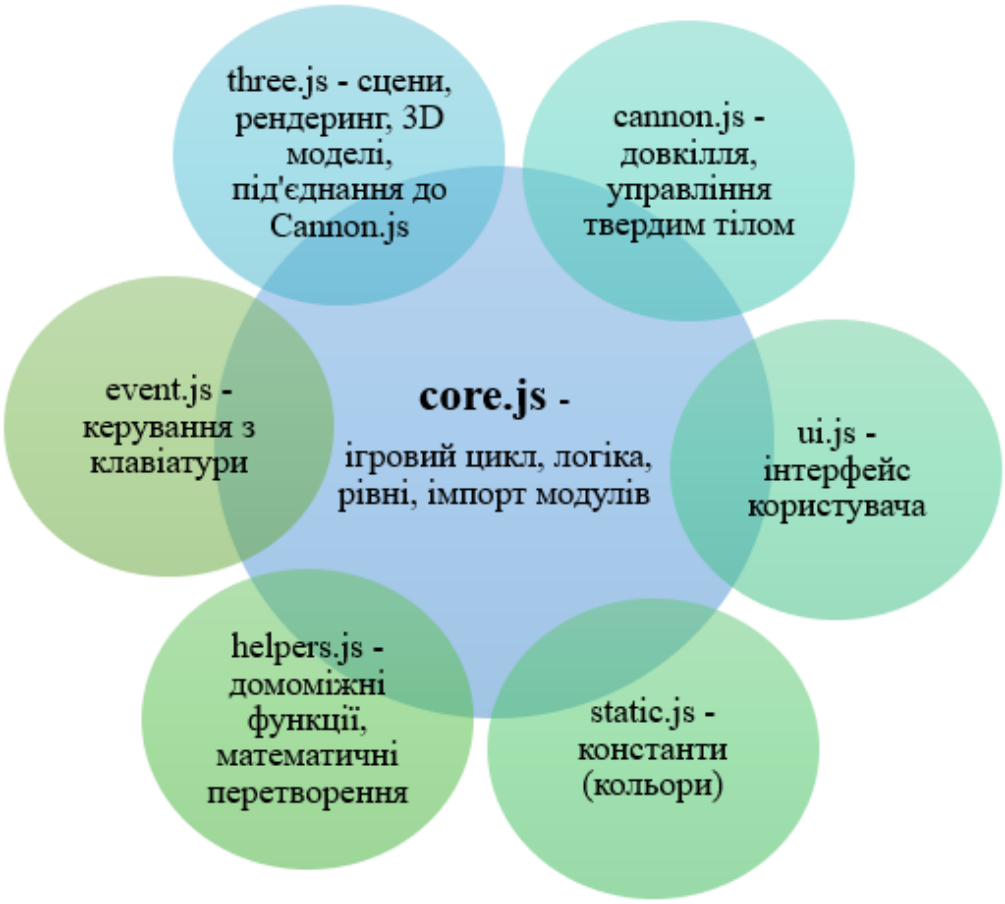


Рис. 3.2.2 – Складові гри

Розглянемо їх детальніше.

```
//THREE.JS

window.game.three = function() {
    var _three = {
        /*Атрибути*/
        // DOM контейнер який буде містити фінальний canvas елемент THREE.js
        domContainer: null,
        // Стиснення розміру камери, щоб обмежити поле зору користувача
        cameraSizeConstraint: null,
        // Сцена, камера і рендерер
        camera: null,
        scene: null,
        renderer: null,
        // Дефолтні налаштування поля зору
        fov: 45,

        /*Методи*/
        init: function() {},
        destroy: function() {},
        setup: function () {},
        render: function() {},
        onWindowResize: function() {},
        createModel: function() {},
        createCannonGeometry: function() {},
        createCannonHalfExtents: function() {}
    };

    return _three;
};
```

У файлі створюється цільовий DOM контейнер і під’єнується до Three.js рендерера, який додає елемент «полотна» у нього. Тут активується камера, приймає деякі параметри для перегляду. Окрім налаштування розміру вікна, представлено і допоміжні методи для Cannon.js, наприклад, обробка тривимірних моделей (використовуючи Blender і JSON формат) та забезпечення правильного відтворення сітки, що важливо для канунівських обмежувальних сфер (використовуються для розрахунку зіткнень) та паралелепіпедів зі сторонами, паралельними до осей координат, що також обмежують геометричний об’єкт у просторі.

```
//CANNON.JS

window.game.cannon = function() {
    var _cannon = {
        /*Атрибути*/
        // Cannon.js довкілля утримує всі тверді тіла на рівні
        world: null,
        // Тіла відповідають фізичним об'ктам всередині Cannon.js
        bodies: [],
        // Visuals  це візуальні зображення тіл зрештою візуалізуються THREE.js
        visuals: [],
        // body count в індексі
        bodyCount: 0,
        // Дефолтні значення тертя та відновлення
        frictionDefault: 0.0,
        restitutionDefault: 0.0,
        // Дефолтна Z gravity (наближено 9,806)
        gravity: -10,
        // Швидкість зміни кроку фізичкого моделювання Cannon.js (інтервали)
        timestep: 1 / 8,
        // Для слоумо
        timestepSlowMotion: 1 / 20,
        // передається у game.core.js
        playerPhysicsMaterial: null,
        // solidMaterial для всіх рівневих об'єктів
        solidMaterial: null,

        // Methods
        init: function() {},
        destroy: function () {},
        setup: function () {},
        overrideCollisionMatrixSet: function() {},
        getCollisions: function() {},
        rotateOnAxis: function() {},
        createRigidBody: function() {},
        createPhysicsMaterial: function() {},
        addVisual: function() {},
        removeVisual: function(){},
        removeAllVisuals: function(){},
        updatePhysics: function() {},
        shape2mesh: function() {}
    };

    return _cannon;
};
```

Тут базуються усі налаштування, що стосуються гравітації, тертя та відновлення, а також усі основні функції для оновлення симуляції фізики.

```
//CORE.JS
window.game.core = function () {
    var _game = {
        // Атрибути
        player: {
            // Атрибути
            speed: 2,
            speedMax: 65,
            rotationSpeed: 0.007,
            rotationSpeedMax: 0.040,
            damping: 0.9,
            rotationDamping: 0.8,
            cameraOffsetH: 280,
            cameraOffsetV: 180,

            // Методи
            create: function() {},
            update: function() {},
            updateCamera: function() {},
            updateAcceleration: function() {},
            processUserInput: function() {},
            accelerate: function() {},
            rotate: function() {},
            jump: function() {},
            updateOrientation: function() {}
        },
        level: {
            // Методи
            create: function() {}
        },

        // Методи
        init: function() {},
        destroy: function() {},
        loop: function() {},
        initComponents: function () {}
    };

    return _game;
};
```

Ядро містить усі інші методи, що відповідають за логіку гри: діяльність гравця, структура рівнів та їхні властивості починаючи від прискорення, обертання гравця, закінчуючи рухом камери.

```
//EVENTS.JS
window.game.events = function() {
    var _events = {
        // Атрибути
        keyboard: {
            // Attributes
            // Використовується у core.player.controlKeys
            keyCodes: {},
            // Зберігає інформацію про натискання клавіш у реальному часу
            pressed: {},

            // Методи
            onKeyDown: function() {},
            onKeyUp: function() {}
        },

        // Методи
        init: function() {}
    };

    return _events;
};
```

Даний файл відповідає за своєчасне реагування системи на натискання контрольних клавіш.

```
//HELPERS.JS
window.game.helpers = {
    // Конвертує полярні координаи в декартові, використовуючи довжину і радіан
    polarToCartesian: function() {},
    // Конвертує радіани в градуси (1 радіан = 57,3 градуса)
    radToDeg: function() {},
    // з градусів у радіани
    degToRad: function() {},
    // Створює випадкове число між фіксованим діапазоном
    random: function() {},
    // Метод використовується в ігровому ядрі для скидання рівня, щоб не потрібно було
    // скидати всі властивості та стани рівнів до їхніх початкових налаштувань
    cloneObject: function() {}
};


//STATIC.JS
    window.game.static = {
    colors: {}
};
```

```
//UI.JS
window.game.ui = function() {
    var _ui = {
        // Attributes
        elements: {
            height: null,
            airboosts: null,
            airboostsProgress: null,
            slowmotion: null,
            infoboxKeyboard: null,
            infoboxGameover: null,
            infoboxHelp: null,
            btnToggleSound: null,
            btnCloseInfoboxKeyboard: null,
            btnCloseInfoboxGameover: null,
            btnCloseInfoboxHelp: null,
            btnShowInfoboxHelp: null,
            btnShowInfoboxKeyboard: null,
            gameoverScore: null,
            gameoverHighestScore: null
        },

        // Methods
        init: function () {},
        destroy: function () {},
        getElements: function () {},
        bindEvents: function () {
            setTimeout(function () {}, //.....всі інші методи, що відповідають за користувацький інтерфейс....
```

Користувацький інтерфейс.

```
//INIT.JS
if (!Detector.webgl) {
    Detector.addGetWebGLMessage();
    document.querySelector("#game-ui").className = "hidden";
} else {
    window.gameInstance = window.game.core();
    window.gameInstance.init({
        domContainer: document.querySelector("#game"),
        cameraSizeConstraint: {
            height: 110
        }
    });
}
```

Цикл гри можна представити у такому простому вигляді:

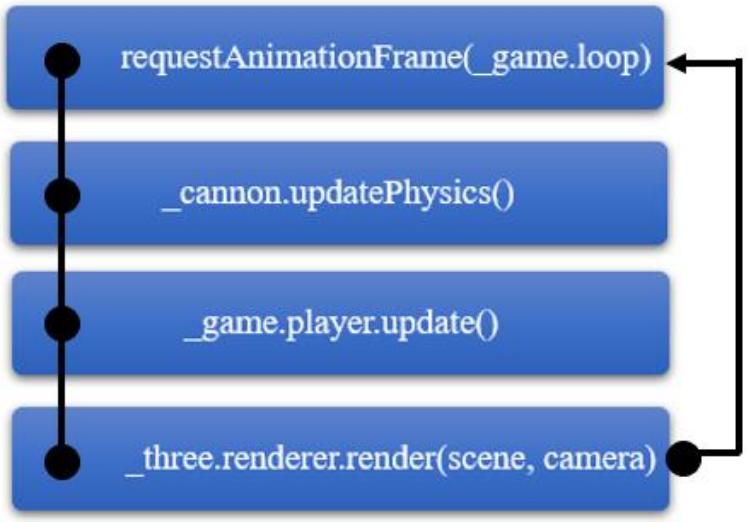


Рис. 3.2.3 – Цикл

3.3 Демонстрація результату

Для цілей демонстрації результату гри ми розробили відео, в якому продемонстровано курсовий проект у вигляді браузерної гри. Відео доступне за посиланням: https://youtu.be/mEigf4cgOi0

У додатку А представлені основні складові гри.

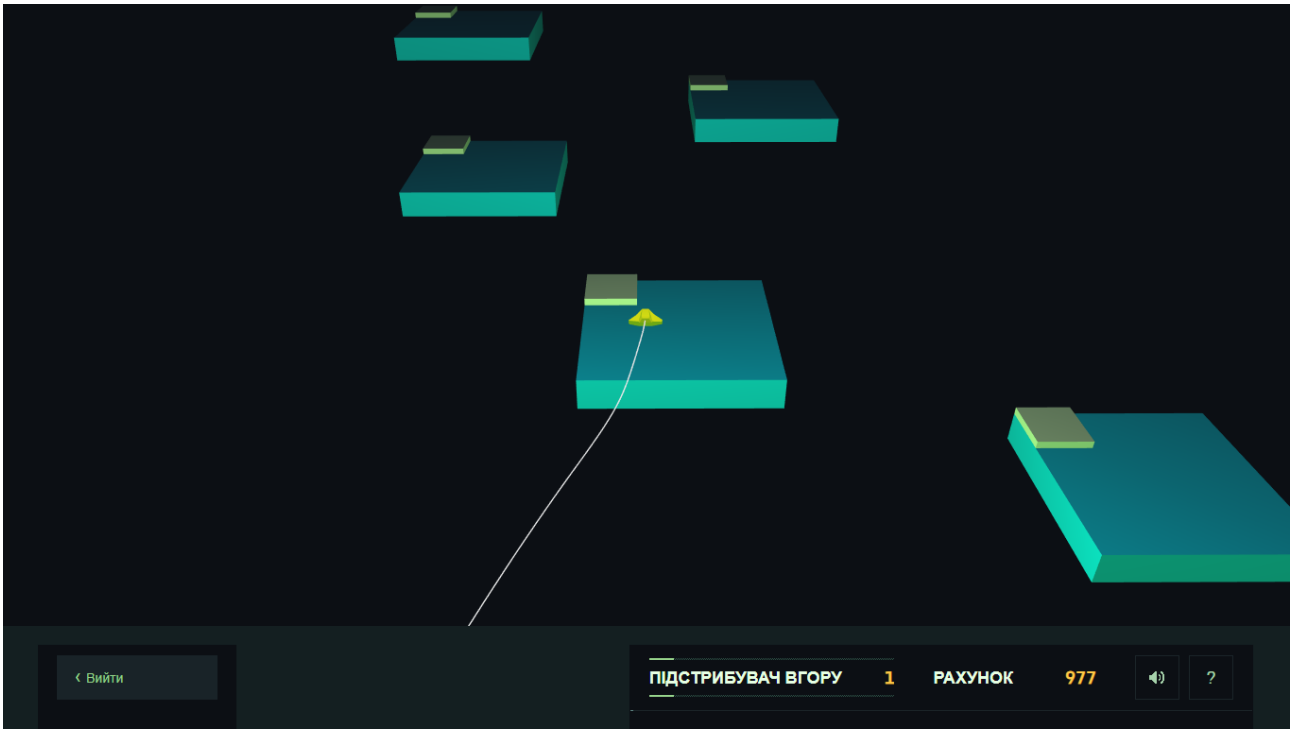


Рис.3.3.1 – Гра BlockJumper

ВИСНОВКИ

Сфера вебпрограмування, зокрема, усе, що пов’язане із браузерною анімацією, тривимірною графікою розвивається швидким темпом завдяки величезну попиту на неї серед користувачів мережі Internet та, відповідно, зі сторони бізнесу. Відповідно, розвиваються засоби розробки, створюються і вдосконалюються бібліотеки, рушії тощо.

Наш проект було розроблено з використанням JavaScript бібліотеки Three.js та Cannon.js. Проект добре структурований, розбитий на основні сегменти, кожен з яких відповідає за конкретний елемент гри: ядро, користувацький інтерфейс, візуалізація сцени, фізика і т.д.

Розроблена нами гра Block Jumper Game є простою у користуванні, однак потребує сконцентрованості та швидкої реакції. Проаналізувавши збірки найпопулярніших онлайн-ігор у додатку Android Play Market, ми можемо зробити висновки, що такий тип ігор є дуже популярним серед користувачів.

Гру на базі WebGL можна відкрити у всіх найвідоміших браузерах – Chrome, Firefox, Safari, Opera, Internet Explorer, Edge. Тобто, вона є доступною користувачам.

Детально вивчаючи й аналізуючи середовище веб-розробки, анімації, тривимірної графіки, а також сферу браузерних ігор, ми прийшли до висновку, що це поле діяльності надзвичайно цікаве для подальшого дослідження, реалізації проектних ідей і задумів.

СПИСОК ПОСИЛАНЬ

[1] https://graphics.cg.uni-saarland.de/courses/cg1-2018/slides/CG01b-History_Applications.pdf

[2] https://graphics.stanford.edu/courses/cs248-05/History-of-graphics/History-of-graphics.pdf

[3] https://ohiostate.pressbooks.pub/graphicshistory/chapter/13-3-evans-and-sutherland/

[4] https://www.etudes.ru/ru/etudes/cat-animation/

[5] https://www.loc.gov/static/programs/national-film-preservation-board/documents/computer_hand2.pdf

[6] https://cg.cs.tsinghua.edu.cn/course/docs/chap5.pdf

[7] https://users.cs.northwestern.edu/~ago820/cs395/Papers/Phong_1975.pdf

[8] https://www.siggraph.org/about/awards/1999-coons-award/

[9] https://medium.com/@arjun07/differences-between-high-poly-vs-low-poly-3d-models-348cab56e82e

[10] JungHyun Han. “3D Graphics for Game Programming”, Korea University, Seoul, South Korea, Taylor and Francis Group, LLC

[11] https://www.sciencedirect.com/topics/engineering/surface-modeling

[12] https://www.embl-hamburg.de/biosaxs/courses/embo2014/slides/rigid-body-svergun.pdf

[13] https://www.khronos.org/webgl/

[14] https://wiki.voxelplugin.com/Main_Page

[15] https://doc.photonengine.com/en-us/pun/current/getting-started/pun-intro

[16] https://docs.godotengine.org/en/3.0/classes/class_spritebase3d.html

[17] https://threejs.org/examples/#webgl_animation_cloth

[18] https://threejs.org/docs/

[19] https://github.com/kripken/ammo.js/

[20] https://schteppe.github.io/cannon.js/docs/

[21] https://github.com/lo-th/Oimo.js/

[22] https://aframe.io/docs/1.0.0/introduction/

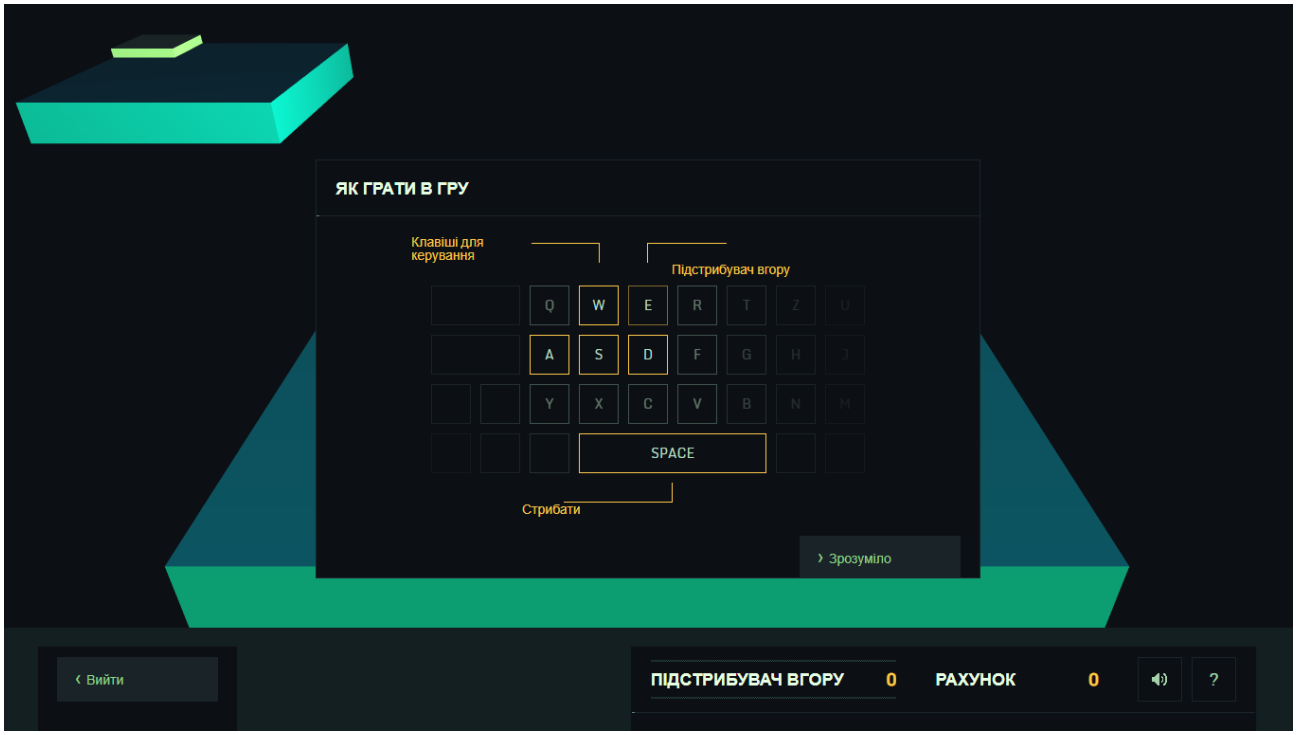
[23] https://doc.babylonjs.com/

[24] Jos Dirksen. “Learning Three.js: The JavaScript 3D Library for WebGL”, Published by Packt Publishing Ltd., 2013

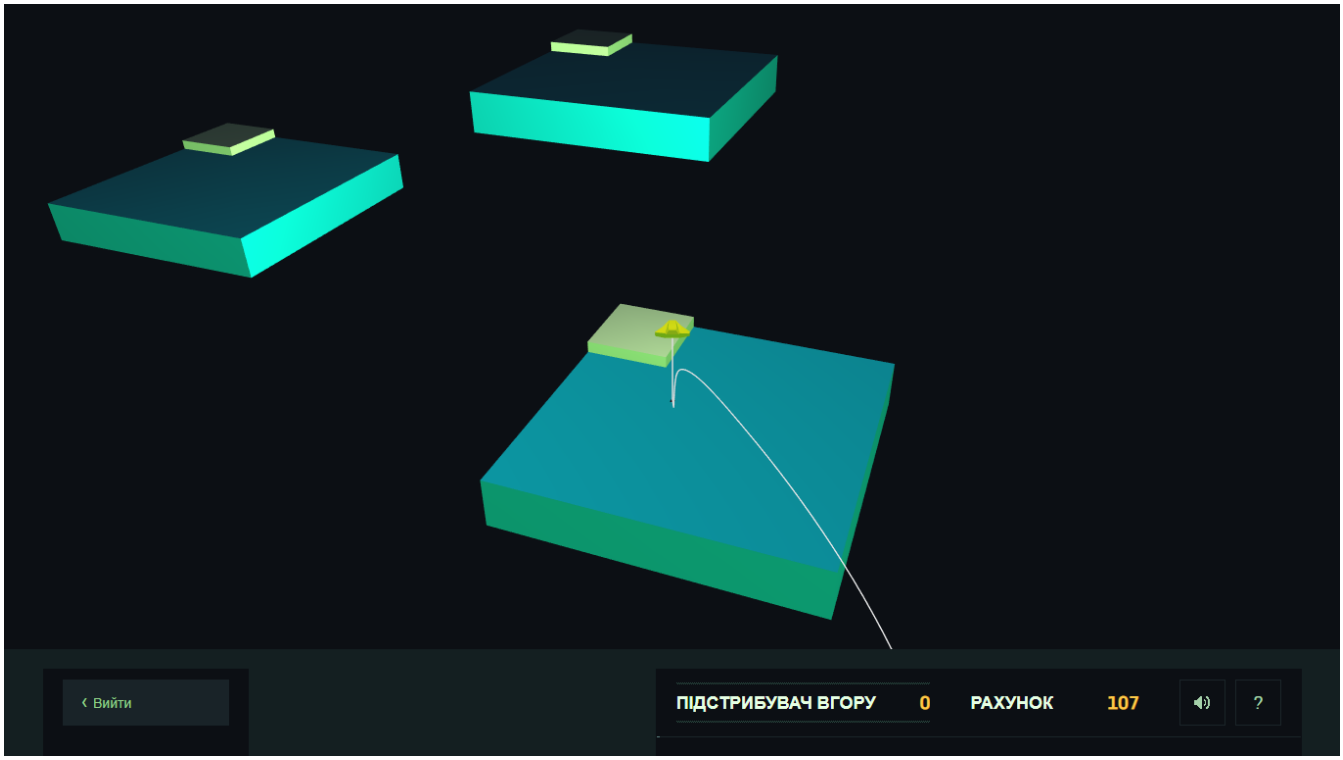
[25] Jos Dirksen. “Three.js Essentials”, Published by Packt Publishing Ltd., 2014

ДОДАТОК А

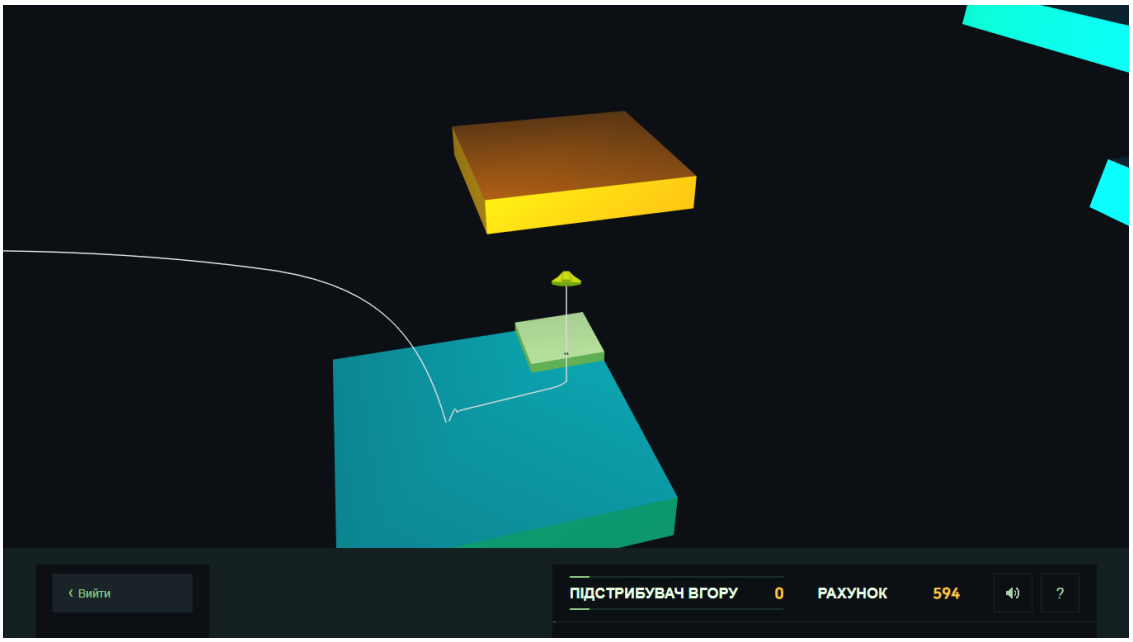
1) Стартова сторінка



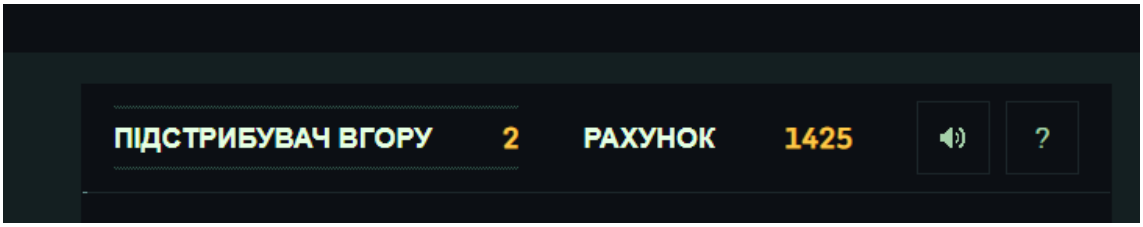
2) Базові блоки



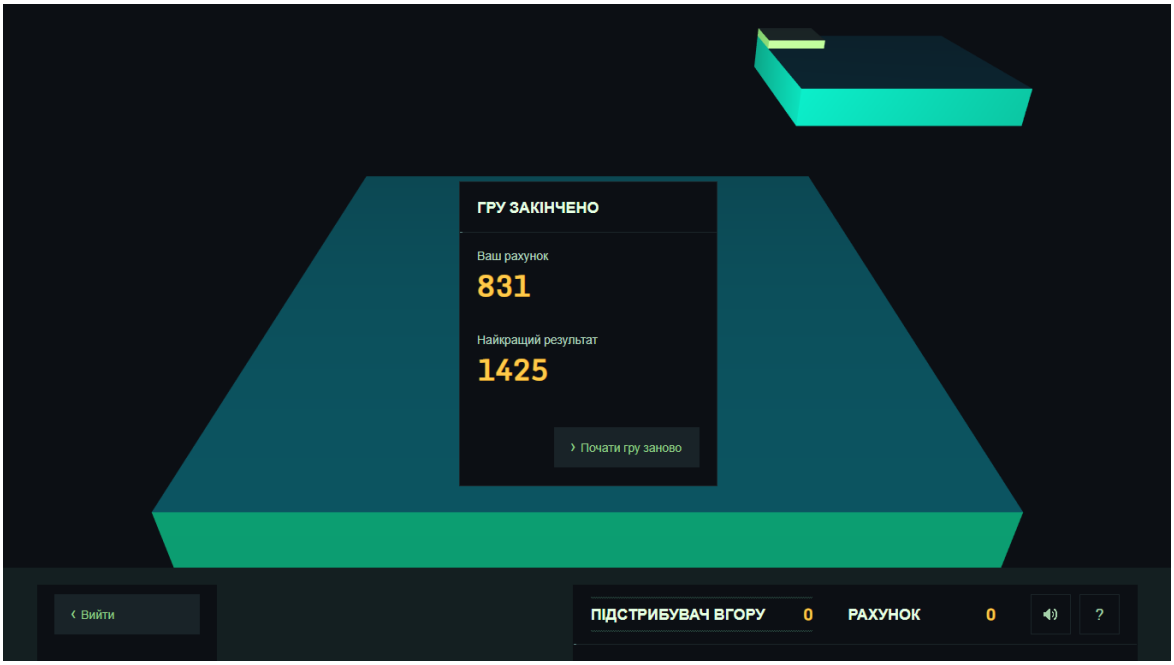
3) Блоки для підсилення стрибка



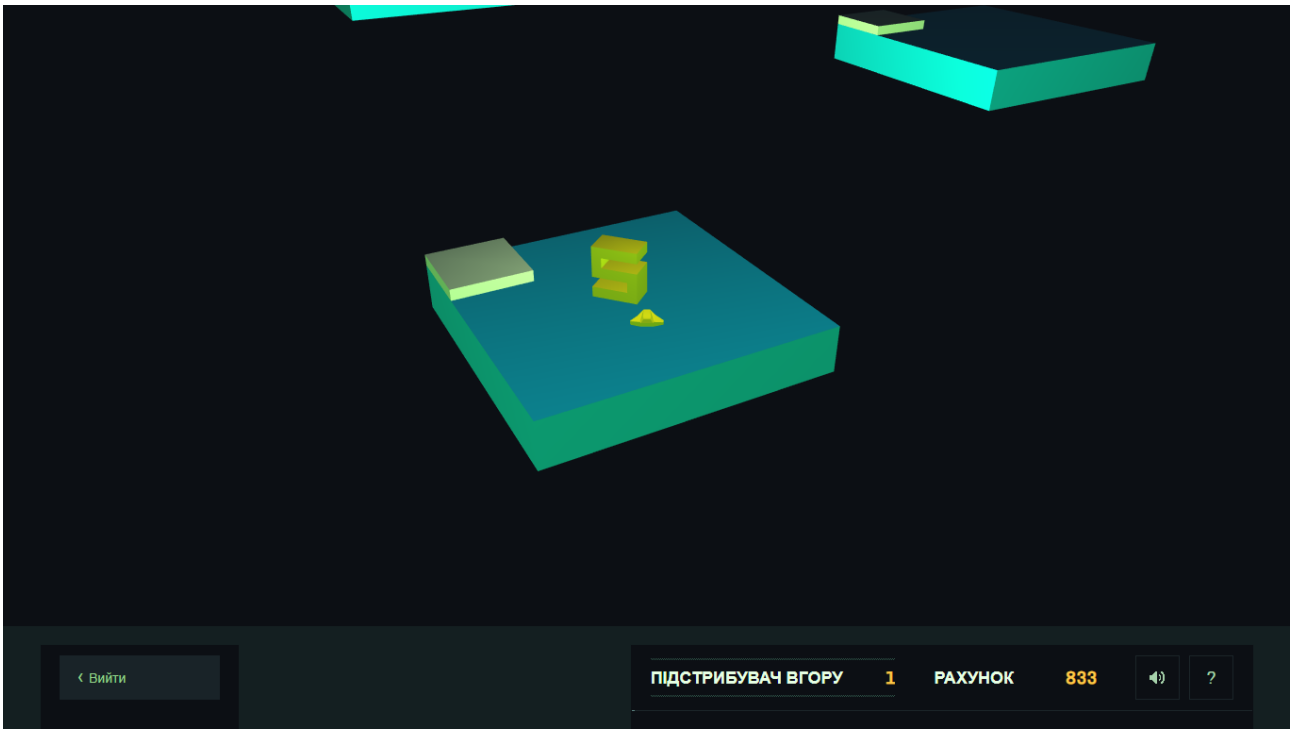
4) Нараховувач «Підстрибувачів вгору» та поточний рахунок



5) Отримання результату гри



6) Slow Motion режим



7) Інформаційна панель