НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет електроніки
(повна назва інституту/факультету)

Кафедра звукотехніки та реєстрації інформації
(повна назва кафедри)

«На правах рукопису»
УДК 004.428.4

«До захисту допущено»
Завідувач кафедри
_____ Г.Г.Власюк
(підпис) (ініціали, прізвище)
“ 09 ” грудня 2019 р.

Магістерська дисертація

спеціальність 171 Електроніка
(код і назва спеціальності)

на тему: «Особливості застосування графічних бібліотек сучасних браузерів»

Виконав: студент VI курсу, групи ДВ-81мп
(шифр групи)

Кучеренко Алевтина Олександрівна _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник професор, д.т.н., професор Розорінов Г.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут (факультет) Факультет електроніки
(повна назва)

Кафедра звукотехніки та реєстрації інформації
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (освітня програма) 171 Електроніка

(Електронні системи мультимедіа та засоби Інтернету речей)

(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Г.Г.Власюк
(підпис) (ініціали, прізвище)
« 10 » вересня 2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Кучеренко Алевтині Олександрівні

(прізвище, ім'я, по батькові)

1. Тема дисертації «Особливості застосування графічних бібліотек сучасних браузерів»

науковий керівник дисертації Розорінов Георгій Миколайович, д.т.н., професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « 07 » листопада 2019р. № 3859-с

2. Строк подання студентом дисертації 09.12.2019 р.
3. Об'єкт дослідження: сучасні графічні бібліотеки.
4. Предмет дослідження (Початкові дані – для магістерської дисертації за освітньо-професійною програмою): особливості застосування різних графічних бібліотек в сучасних браузерах, методи та технології, середовище розробки програмного забезпечення JetBrains WebStorm, бібліотека ThreeJS, комп'ютер на базі Windows, браузер Google Chrome 78.0.3904.108.

5. Перелік завдань, які потрібно розробити: 1) Проаналізувати роботу сучасних веб браузерів та дослідити базові алгоритми створення графіки в браузерах; 2)Розглянути існуючі методи реалізації графіки та відтворення анімації; 3)Проаналізувати ThreeJS та його можливості; 4) Привести максимально зручний та простий приклад використання графічної бібліотеки для користувачів.

6. Перелік графічного (ілюстративного) матеріалу: 1) 64 рис, 25 табл., 1 презентація, 8 слайдів.

7. Орієнтовний перелік публікацій: 1.) Оптимізація пам`яті та оптимізації продуктивності WebGL // Міжнародний електронний науковий журнал «Наука онлайн». Випуск 12/2019, 2019. 2.) WebGL, вимірювання продуктивності та методи оптимізації WebGL – додатків в браузері // Міжнародний електронний науковий журнал «Наука онлайн». Випуск 12/2019, 2019.

8. Дата видачі завдання 10. 09. 2018 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Написання першого розділу: Аналітичний огляд сучасних браузерів та методів побудови графічних моделей та інтерфейсів	16.04.2019	
2	Написання другого розділу: Програмні засоби створення графічних моделей	14.06.2019	
3	Написання третього розділу: Особливості застосування WebGL. Порівняння графічних бібліотек	10.09.2019	
4	Написання четвертого розділу: Розробка тривимірної гри з використанням графічної бібліотеки Three.JS	15.10.2019	
5	Підготовка матеріалів до друку та оформлення пояснювальної записки	30.11.2019	
6	Підготовка та оформлення презентації для доповіді	03.12.2019	

Студент _____ А.О. Кучеренко
(підпис) (ініціали, прізвище)

Науковий керівник _____ Г.М. Розорінов
(підпис) (ініціали, прізвище)

УДК 004.428.4

РЕФЕРАТ

Магістерська дисертація: 125 с., 64 рис., 25 табл., 1 дод., 29 джерел.

ГРАФІЧНІ БІБЛІОТЕКИ, WEBGL, THREE JS, BABYLON JS, БРАУЗЕР, ВІЗУАЛІЗАЦІЯ.

Актуальність теми роботи полягає в тому, що використання графічних бібліотек набуло широкого розповсюдження у сучасних сферах діяльності, таких як кіновиробництво, реклама, онлайн трансляція подій тощо. Це обумовлює розвиток графічних технологій.

Метою дослідження є порівняння особливостей використання графічних бібліотек в браузерах. Визначити найкращі варіанти реалізації в плані швидкості роботи, швидкості створення та загального результату для подальшого застосування в кіновиробництві та в ігровій індустрії.

Для досягнення поставленої мети необхідно виконати такі завдання:

- проаналізувати роботу сучасних веб браузерів;
- дослідити базові алгоритми створення графіки в веб браузерах;
- розглянути існуючи приклади реалізації графіки та відтворення анімації;
- провести аналіз особливостей застосування графічних бібліотек;
- виконати порівняльний аналіз бібліотек, виявити найвигідніший варіант реалізації графіки, привести максимально зручний та простий приклад використання для користувачів.

Об’єкт дослідження – сучасні графічні бібліотеки.

Предмет дослідження – особливості застосування різних графічних бібліотек в сучасних браузерах.

Методи дослідження – теоретичне дослідження роботи сучасних веб браузерів, базових алгоритмів створення графіки, прикладів реалізації та відтворення анімацій, аналіз особливостей застосування готових систем створення графічних моделей та реалізації в веб браузерах, критичний аналіз

існуючих технологій.

Наукова новизна одержаних результатів: запропоновано підхід до створення графічних інтерфейсів, що забезпечує швидкодію та якісні результати, спрощений алгоритм, максимально зрозумілий для звичайних користувачів та який можна реалізувати для навчальних потреб.

Практична цінність одержаних результатів: на основі аналітичного та практичного аналізу **з**апропоновано підхід до створення графічних інтерфейсів. Використання Three js бібліотеки у розробці графічних інтерфейсів має переваги у швидкості та якості розробки на відміну від всіх інших розглянутих технологій. Запронований підхіх не вимагає поглиблених знань OpenGL або WebGl, має можливості інтеграції з сучасними програмами для створення графічних моделей та може бути маштабованим в майбутньому.

Публікації:

1. А.О. Кучеренко. «Оптимізація пам’яті та оптимізації продуктивності WebGL»./Кучеренко А.О. Міжнародний електронний науковий журнал «Наука онлайн». Випуск 12/2019, 2019.
2. А.О. Кучеренко. «WebGL, вимірювання продуктивності та методи оптимізації WebGL - додатків в браузері» / Кучеренко А.О. Міжнародний електронний науковий журнал «Наука онлайн». Випуск 12/2019, 2019.

SUMMARY

Master’s dissertation: 125 p., 64 pic., 25 tabl., 29 sources.

GRAPHIC LIBRARIES, WEBGL, THREE JS, BABYLON JS, BROWSER, VISUALIZATION.

Actuality of work is that the use of graphic libraries has become widespread in modern fields of activity, such as film production, advertising, online events. This causes the development of graphic technologies.

The object of study is modern graphic libraries.

The purpose of the study is to compare the features of using libraries in browsers. Identify the best implementation options in terms of speed, creation speed, and overall output for future use in the film industry and the gaming industry.

To achieve this goal, you must perform the following tasks:

– analyze the work of modern web browsers;

– explore basic algorithms for creating graphics in web browsers;

– consider existing examples of graphics implementation and animation playback;

– to analyze the features of the use of graphic libraries;

– to perform a comparative analysis of these libraries, to identify the most advantageous variant of implementation of graphics, to give the most convenient and simple example of use for users.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API	–	Application Programming Interface (Прикладний програмний інтерфейс);
CPU	–	Central processing unit (центральний процесор) ;
GPU	–	Graphics processing unit (графічний процесор) ;
WebGL	–	Web-based Graphics Library(Веб-бібліотека графіки) ;
HTML	–	Hypertext Markup Language(Мова розмітки гіпертекстових документів) ;
CSS	–	Cascading Style Sheets(Каскадні таблиці стилів) ;
DOM	–	Document Object Model (Об'єктна модель документа) ;
3D	–	3D (Тривимірний опис об'єкта) ;
SVG	–	Scalable Vector Graphics (масштабована векторна графіка) ;
CDN	–	Content Delivery Network (Мережа розповсюдження контенту) ;
PC	–	Personal computer (Персональний комп'ютер);
XML	–	Extensible Markup Language (Розширювана мова розмітки);

ВСТУП

Веб-технології міцно увійшли в повсякденне життя. Кожен проводить у всесвітній павутині досить велику кількість часу - перегляд новин, здійсненння покупок, спілкування та робота. Індустрія послуг та розваг в мережі Інтернет стрімко розвивається, провідні розробники програмного забезпечення покращують підтримку тривимірної графіки в своїх продуктах.

Традиційно підтримка графіки обмежувалася високопродуктивними комп'ютерами або спеціалізованими ігровими консолями, а програмування вимагало застосування складних алгоритмів. Однак завдяки зростанню продуктивності персональних комп'ютерів і розширенню можливостей браузерів стало можливим створення і відображення тривимірної графіки із застосуванням веб-технологій.

На відміну від інших технологій для роботи з тривимірною графікою WebGL призначена для використання в веб-сторінках і не вимагає установки спеціалізованих розширень або бібліотек. Одна з переваг – додатки, які конструюються як веб-сторінки, тобто одна і та ж програма буде успішно виконуватися на самих різних пристроях в тому числі на смартфонах, планшетних комп'ютерах та ігрових консолях. Це означає, що WebGL буде надавати все більш великий вплив на співтовариство розробників і стане одним з основних інструментів програмування графіки.

З розвитком HTML розробники отримали можливість створювати все більш складні веб-додатки. На початку свого розвитку мова HTML пропонувала тільки можливість відображення статичного контенту, але з додаванням підтримки JavaScript стало можливим реалізовувати більш складні взаємодії елементів і відображення динамічного контенту. Впровадження стандарту HTML5 дозволило використовувати нові можливості, включаючи підтримку двомірної графіки у вигляді тега canvas. Створення технології WebGL дозволило відображати і маніпулювати тривимірною графікою на веб-сторінках за допомогою JavaScript.

За допомогою WebGL розробники можуть створювати абсолютно нові інтерфейси, тривимірні ігри і використовувати тривимірну графіку для візуалізації різної інформації. Незважаючи на значні можливості, WebGL відрізняється від інших технологій доступністю і простотою використання, що сприяє її швидкому поширенню.

Сучасні компанії прагнуть до реалістичності своєї графіки, використовуючи найновіші технології. Тому доцільно дослідити особливості реалізації існуючих засобів створення.

1. АНАЛІТИЧНИЙ ОГЛЯД СУЧАСНИХ БРАУЗЕРІВ ТА МЕТОДІВ ПОБУДОВИ ГРАФІЧНИХ МОДЕЛЕЙ ТА ІНТЕРФЕЙСІВ.

1.1. Сучасні браузери

Браузер - це спеціально розроблена програма призначена для перегляду веб-сайтів. Самі сайти по своїй суті являють собою певний набір коду , браузер же, призначений для того, щоб цей самий код перетворити в те, що відображається на екрані.

Перший браузер, який мав графічний інтерфейс, тобто не тільки просто текст на чорному фоні, був розроблений в 1993 році і мав назву NCSA Mosaic. Саме він послужив основою для створення інших веб-оглядачів, так як, розробники свого часу відкрили його вихідний код для всіх бажаючих. Так, на основі NCSA Mosaic був розроблений найпопулярніший свого часу браузер Netscape Navigator, сталося це в 1994 році, він мав приголомшливий успіх і приносив непоганий прибуток компанії його розробника. Хочеться відзначити, що внутрішнім ім'ям Netscape Navigator було – Mozilla.

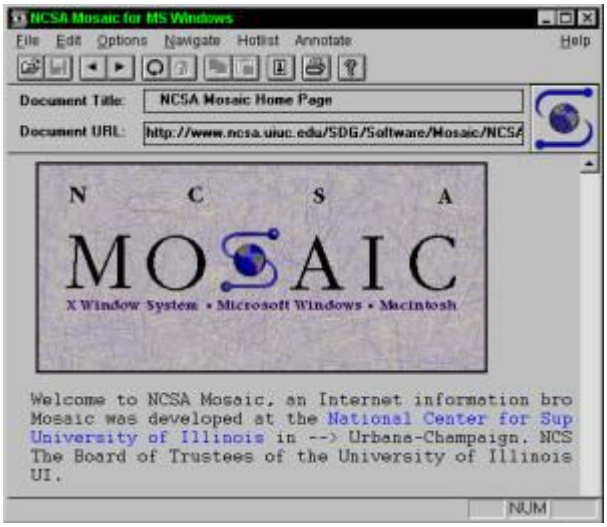

Рисунок 1.1. Браузер Mosaic.

Рисунок 1.2. Браузер Navigator.

Компанія Microsoft не могла не помітити такий успіх Netscape Navigator і розробила свій власний браузер в 1995 році, так само зроблений на основі NCSA Mosaic. Назву йому дали - Internet Explorer. Внаслідок саме Internet Explorer (IE) став невід'ємною частиною всіх операційних систем цієї компанії. Так, як ОС Windows користувалося величезна кількість користувачів, IE швидко завоював цю нішу і завоював близько 95% всього ринку. Це і призвело до закриття проекту Netscape Navigator, адже конкурувати з такою монополією було неможливо.

Рисунок 1.3. Перша версія IE.

Netscape купує компанія AOL Time Warner, яка робить вихідний код Navigator відкритим. Далі AOL, в зв'язку зі своїм закриттям, передає всі права і свої розробки в нову компанію - Mozilla Foundation, яка продовжила розвивати їх ідеї.

У 1996 році з'явилася Opera, яка, завдяки невеликій вазі і швидкої завантаженні сторінок, стала в той час найпопулярнішою альтернативою Internet Explorer по всьому світу.

Internet Explorer ж став втрачати свої позиції, тому що Microsoft не оновлював його аж до жовтня 2006 року, тому що у них була завойована і так велика частина ринку. Але Internet Explorer був до цього часу вже настільки глючний і мав безліч дірок в безпеці, що з часом став одним з найменш улюблених і не популярних браузерів - так триває і до нинішнього моменту, не дивлячись на появу його нових, набагато поліпшених версій.

У листопаді 2004 року з'явився улюблений багатьма веб-оглядач Mozilla Firefox, який грунтувався на проекті Mozilla Suite.

У 2006 році компанія Apple випустила свій продукт під назвою Safari, а в 2008 році на ринок вступила і компанія Google, випустивши Google Chrome.

До сьогоднішнього дня було створено і випущено безліч різних цікавих веб-оглядачів - кожен з них має свої унікальні функції і фішки.

1.1.1. Приклади та порівняння браузерів.

Google Chrome

Рисунок 1.4. Браузер Google Chrome.

Одна з найбільш швидких, зрозумілих і популярних програм такого роду. Даний проект розробляє корпорація - гігант Google. Chrome мало не самий передовий браузер з усіх, оновлення виходять практично щотижня і включають в себе завжди найкращі і передові інновації. Саме на основі Chrome зараз робляться багато інших веб-оглядачі, наприклад такі як, Opera, Yandex.Browser, Orbitum і

т.д. Славиться величезною кількістю програм, розширень, тем та ігор в своєму маркеті.

Mozilla Firefox

Рисунок 1.5. Браузер Mozilla Firefox.

Не менш популярний браузер, ніж Google Chrome, так само швидкий і красивий. Має свій унікальний інтерфейс і можливість розширення функціоналу за рахунок установки безлічі різних розширень. Саме Firefox свого часу зламав хід історії і поламав монополію Internet Explorer на ринку. Програма працює на всіляких операційних системах і постійно оновлюється.

Opera

Рисунок 1.6. Браузер Opera.

Один з найбільш загальновідомих і затребуваних браузерів в країнах СНД. Опера з'явилася більше 20 років тому, і, в процесі свого розвитку, змінила повністю движок на якому вона до цього розроблялася. Opera володіє зрозумілим і красивими інтерфейсом, дуже великою швидкістю роботи, можливістю встановлення різних розширень і віджетів. Програма постійно оновлюється і має безліч своїх унікальних фішок. Має в своєму арсеналі функцію «Turbo», яка значно зменшує час завантаження сторінок.

Safari

Рисунок 1.7. Браузер Safari.

Браузер, який спеціально розроблявся для операційної системи Mac OS, корпорацією гігантом Apple. Свого часу була випущена його версія і для ОС Windows, але, на жаль, з 2012 року нові версії для цієї ОС більше не виходили. Сафарі володіє інноваційними технологіями, унікальним інтерфейсом і дуже великою швидкістю роботи. Так само, відмінною його особливістю є «глянсовий інтерфейс».

Internet Explorer

Рисунок 1.8. Браузер Internet Explorer.

Цей браузер розробляється компанією Microsoft і поставляється вже встановленим в систему Windows. Після провалу версії Internet Explorer 6, IE знайшов саму негативну сторону і в кінці кінців Microsoft вирішила відмовитися від подальших його розробок. Останньою версією є Internet Explorer 11. У Windows 10 Microsoft, в зв'язку з нелюбов'ю користувачів, вирішила припинити його подальшу розробку і випустила вже повністю новий проект під назвою Microsoft Edge.

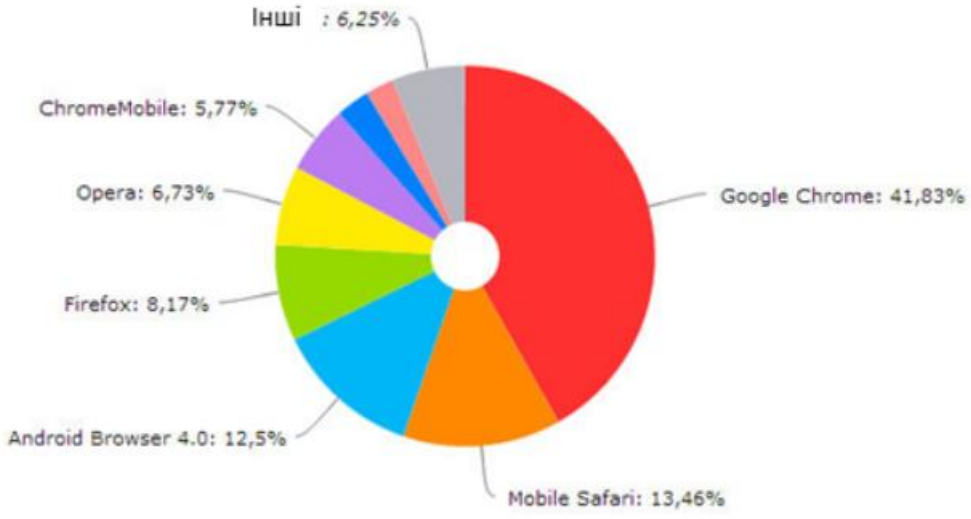

Рисунок 1.9. Графік популярності браузерів по світу.

1.2 HTML5 та CSS3

HTML - це мова розмітки. Він необхідний для структурування веб-документів і реалізації їх в інтернеті. Однак HTML здатний працювати тільки зі структурою всіх даних, щоб визначити як буде виглядати подібна структура, необхідні таблиці стилів або CSS - мова, що використовує розмітку для опису зовнішнього вигляду створеного документа.

HTML5 є останньою зміненої специфікацією HTML, яка надає деякі додаткові мітки і функції (кроссбраузерність, відео, аудіо, анімація і багато іншого), які здатні дати користувачеві більше можливостей для використання різних передових технологій. Одна з таких передових особливостей - Canvas.

Canvas - полотно. Також як професійні художники пишуть складні картини, Canvas дозволяє створювати речі, які раніше було неможливо зробити. Використання 2D форм і образів стало набагато простішим і це все завдяки Canvas.

Показ будь-якої анімації або інтерактивності в межах елемента Canvas майже завжди зачіпає JavaScript. Однак JavaScript, як правило, вимагає значної завантаженості процесора при відтворенні елементів анімації.

Отже, HTML5 не є інструментом для розробки контенту, дизайну, відео або анімації. Це платформа, що дозволяє робити різні речі, використовуючи всі свої інструменти.

Через те, що постійно розвиваються технології сучасні браузери підтримують анімацію та інтерактивний контент без необхідності установки додаткових плагінів, таких, як Flash. Замість цього браузери використовують комбінацію JavaScript, CSS3, HTML5.

Незважаючи на те, що в даний момент вже придумані спеціальні JS-фреймворки, веб-розробники все ж вважають за краще використовувати в своїй роботі програми з багатими візуальними редакторами для роботи з html кодом. Дані інструменти досить схожі між собою, проте це тільки на перший погляд. Такі інструменти використовують найрізноманітніші технології.

CSS-анімація - це можливість зробити веб-сторінку інтерактивною і додати їй певної привабливості.

Властивість transition можна перевести, як перехід, тобто при анімації за допомогою цієї css-властивості протягом деякого часу здійснюється плавний перехід від початкового до кінцевого значення обраних для цього властивостей.

Більшість браузерів на даний момент підтримують властивість CSS3 transition. Виняток становлять Internet Explorer 6-9 і всі версії Opera Mini.

Рисунок 1.10. Статистика підтримки CSS-анімацій браузерами.

Hover-ефекти за допомогою властивості transition

Найчастіше властивість transition застосовується для створення hover-ефектів, або ефектів при наведенні курсору миші. Його суть полягає в тому, що воно прораховує зміни між властивостями, заданими для звичайного стану елемента і при наведенні на нього курсора миші, яке задається за допомогою псевдокласу: hover. Це дозволяє створювати різні красивості від плавної зміни кольору тексту і фону на кнопках-посиланнях до напливають блоків і змінюваних картинок. Дуже цікаві ефекти можна отримати, якщо використовувати псевдоелементи : before і / або : after.

1.2.1 Графіка та анімації.

Властивість transition є універсальною, що означає складається з ряду властивостей, які можна задавати окремо.

Властивості transition:

- transition-delay
- transition-property
- transition-timing-function
- transition-duration

Властивість transition-property задає назву css-властивості для анімації переходу. Оскільки потрібно прорахувати різницю між початковим і кінцевим значенням властивості, то і сама властивість має бути розрахована. Наприклад, можна змінити border-width з 1px до 6px, але не можна перетворити border-style з solid в dashed. Також можна зменшити прозорість елемента за допомогою властивості opacity від значення 1 до 0, але неможливо анімувати перехід від властивості visibility: visible до властивості visibility: hidden або від display: block до display: none. Для цього краще скористатися методами show () або hide () бібліотеки jQuery.

Властивість transition-duration встановлює проміжок часу за який виконується transition-property. Власне, цих двох властивостей цілком достатньо для створення анімації типу transition.

Властивість transition-duration вимірюється або в секундах (1s, 1.5s, 0.8s, .5s) або в мілісекундах (1000ms, 1500ms, 800ms, 500ms). За замовчуванням ця властивість має значення 0, тобто час на анімацію фактично немає, тому, якщо ви забудете вказати це значення, то перехід властивостей не відбудеться.

Властивість transition-timing-function задає тимчасову функцію, яка описує спосіб розрахунку швидкості переходу властивостей html-елемента від одного значення до іншого. За замовчуванням властивість має значення ease, тобто анімація відбувається з деяким уповільненням.

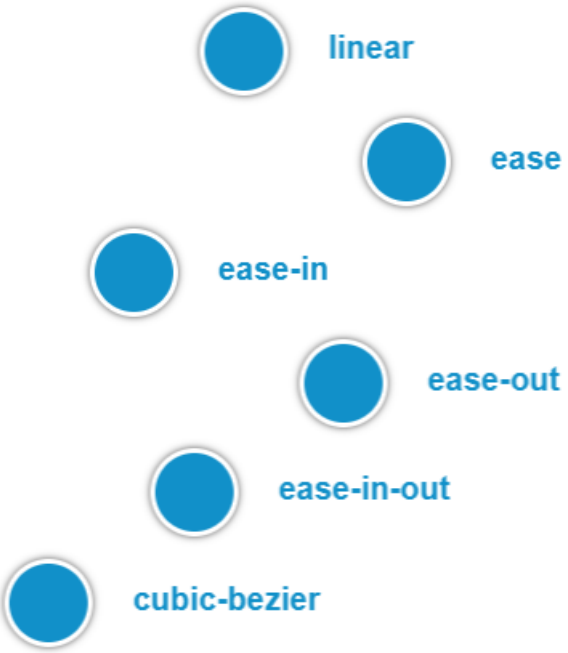

Рисунок 1.11. Варіанти властивості transition-timing-function.

Властивість transition-delay дозволяє вказати затримку в секундах або мілісекундах, після якої буде запущена анімація. Його необов'язково використовувати для всіх анімацій і навіть необов'язково вказувати, тому що за замовчуванням transition-delay дорівнює 0s.

Цю властивість зручно використовувати, коли необхідно, щоб анімація "зіграла" не відразу, а з деяким відставанням.

1.3 Canvas

1.3.1 Історія, можливості та особливості.

Canvas - HTML5 елемент для створення растрового двомірного зображення. Як правило застосовується разом з javascript. Застосовується для відображення графіків і ігрових складових в браузер них іграх, вставки відео, створення повноцінного програвача. Canvas ще застосовується в WebGL для апаратного прискорення 3D-графіки.

Початок відліку блоку розташовується зліва зверху. Від нього і грунтується кожен елемент блоку. Величина місця координат не в обов'язковому порядку відображає величину фактичної інформації, що відображає площу. За замовчуванням його ширина дорівнює 300 пікселям, а висота 150 пікселям.

Вперше canvas елемент був представлений фірмою Apple для Mac OS X з метою його застосування в додатках Dashboard і Safari. На сьогоднішній день canvas застосовується для зведення графіків, нескладних анімацій та ігор в браузерах. Організація Mozilla веде проект під назвою 3D Canvas, метою якого вважається додати низкорівневу допомогу графічних прискорювачів для відображення тривимірних зображень крізь HTML-елемент Canvas. В одному ряду з даними є бібліотеки, реалізують роботу з тривимірними моделями, серед них є Three.

Canvas застосовується, як правило, для малювання графіків для заміток і ігрового поля в деяких браузерних іграх. Але ще має можливість застосовуватися для вбудовування відео в сторінку і створення повноцінного програвача. Canvas має можливість ускладнити завдання робіт по розпізнаванню Капчі. При застосуванні Canvas з сервера завантажується не малюнки, а комплект точок (або метод промальовування), за якими браузер промальовує ілюстрацію (капчу).

Canvas дозволяє розташувати на полотні: зображення, відео, слово. Залити все це суцільним кольором, або ж обвести контури або ж в тому числі і додати градієнт. Додавання тіней подібних на якості css3 box-shadow і text-shadow. І, врешті-решт, зведення фігур з підтримкою вказівки точок.

У разі, якщо немає потреби перемальовувати картину, але потрібно проводити маніпуляції з ним, то ви зможете «сфотографувати» всю картину і зберегти в змінну. У разі якщо зобов'язаний оновлюватися не зображення, а лише тільки його частка, то ви зможете стягувати конкретну зону на полотні і зображати її заново. Браузери можуть поліпшити анімації, які йдуть в один і той же час, зменшивши кількість reflow і repaint, що призведе до збільшення точності анімації. Плюс до всього, в разі якщо проводиться мультиплікація в табі, який невидимий, браузери не стануть продовжувати перемальовування, власне що

призведе до найменшого застосування CPU, GPU, пам'яті і, як наслідок, знизить витрата батареї в мобільних пристроях. Для цього користуйтеся requestAnimationFrame. Всі нинішні браузери мають фільтр розмиття зображення при його підвищенні. Його йде по стопах застосувати, у разі якщо ви нерідко попіксельно обробляєте ілюстрацію. Методом скорочення малюнки, наприклад, в 2 рази і подальшого апаратного нарощування її з підтримкою фільтра. У разі якщо гра дозволяє порізно обробляти поле і складові гри, то містить значення влаштувати 2 полотна приятель над іншому. Для чищення канви найкращим засобом стане впровадження clearRect, втім, в разі якщо чистити лише тільки потрібні ділянки, то швидкість виросте ще більше.

1.3.2 Переваги і недоліки HTML5.

Переваги:

- На відміну від SVG значно зручніше володіти великою кількістю елементів;
- Містить апаратне прискорення.
- Можливо маніпулювати будь-яким пікселем.
- Можливо використовувати фільтри обробки зображень.
- Є велика кількість бібліотек.

Недоліки:

- Навантажує мікропроцесор і оперативну пам'ять.
- Через обмежень збирача сміття, немає можливості очистити пам'ять.
- Потрібно обробляти дії з об'єктами.
- Погана продуктивність при великому розширенні.
- Доводиться зображати порізно кожен елемент.

1.3.3 Порівняння з аналогами. Flash. SVG.

Порівняння Canvas і Flash

- Підтримка передовими браузерами.
- Файловий розподіл і структура.
- Мобільні пристрої.
- Ігри.

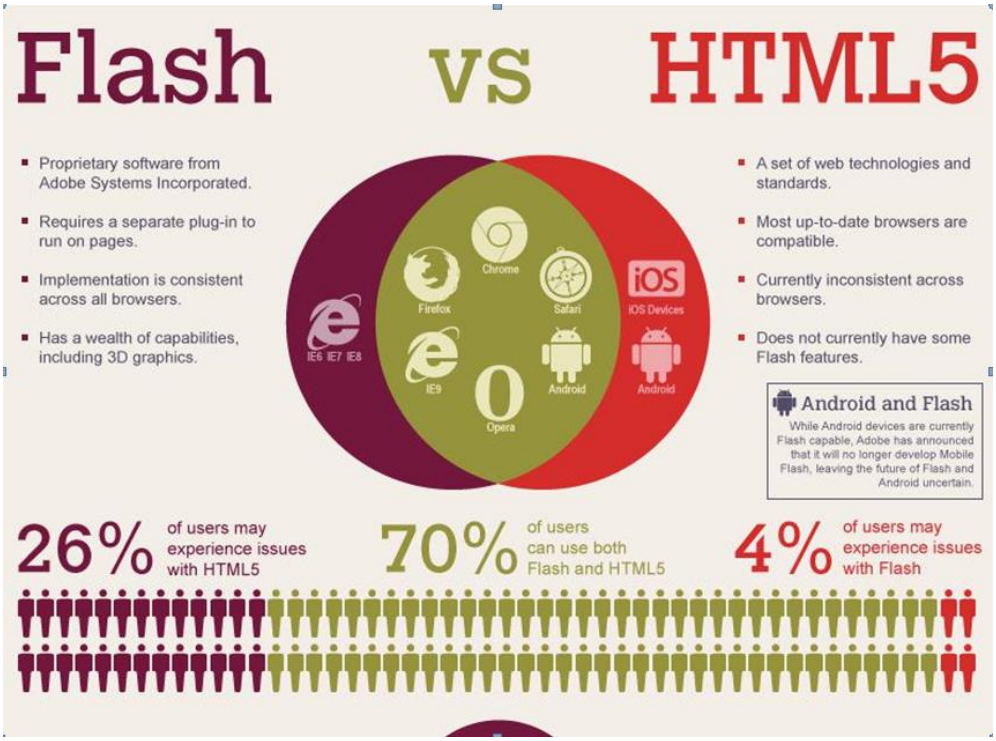

Рисунок 1.12. Flash vs HTML5.

Підтримка передовими браузерами.

На цей момент, 99% десктопних браузерів підтримують Flash Player, і 82% підтримують HTML 5 Canvas.

Файловий розподіл та структура.

Для Flash технології стане необхідним раз файл з розширенням swf. Це досить важливо для ігор і комфортно для заощадження або ж транспортування. Власне що стосується HTML5, то по самій природі HTML трапляється поділ коду в різні файли. В наслідок цього треба бути обачним з їхнім місцезнаходженням та

зазначенням правильного шляху до них. Але, з іншого боку, будь-який окремий файл робить власну функцію, і при редагуванні частини коду є ймовірність встановити виправлення, не турбуючись за цілий план в цілому. Ще сама конструкція свіжого зразка дозволяє пошуковим системам заглядати всередину зроблених з його підтримкою веб та додатків, чого не можна сказати про Flash.

Мобільні пристрої.

З етапу релізу HTML5 однією з ключових переваги мови є її дієздатність працювати на мобільних пристроях. Власне що майже всі Android і iOS-пристроїв не підтримують Flash (97% мобільних браузерів підтримують HTML5 Canvas, а в Flash даний показник 0%), дозволяє застосувати його лише тільки на PC - в 2013 році впав на 10% в порівнянні з 2012. За інформацією бюро інтерактивної реклами, буквально будь-який 2 мешканець USA вважається власником мобільного телефонного апарату з доступом в онлайн, і будь-який 5-ий перегляд веб-сторінок трапляється з мобільного приладу. Цифри зростають щомісяця, і фірми, що формують інтерактивні відео з впровадженням технології Flash, механічно втрачають гігантську аудиторію, яка могла б переглядати цей текст на власних телефонах і планшетах. У наданому пункті здобув перемогу HTML5.

Ігри.

За інформацією блога Digital Buzz, 32% часу, витраченого користувачами мобільних приладів, йде на ігри. З підтримкою HTML5 розробники зуміють робити іграшки, які будуть працювати на всіх пристроях.

Таблиця 1.1. - Порівняння векторної та растрової графіки у різних браузерах

Браузер	Вектор, HTML5 (fps)	Вектор, Flash (fps)	Растр, HTML5 (fps)	Растр, Flash (fps)
Chrome 14.0	9	18	37	14
Firefox 6.0.2	11	17	48	14
IE 9.02	10	20	47	16
Opera 11.51	16	19	9	15
Safari 5.1	2	19	12	15

Порівняння Canvas та SVG

Обираючи між Canvas та SVG, слід враховувати наступні характеристики: характер вирішуваної проблеми та продуктивність .

Canvas слід використовувати для обробки зображень і відео.

SVG слід застосовувати для документів з високою точністю та статичних зображень.

Обидві технології, можна застосовувати для відтворення діаграм, графіків та іграшок.

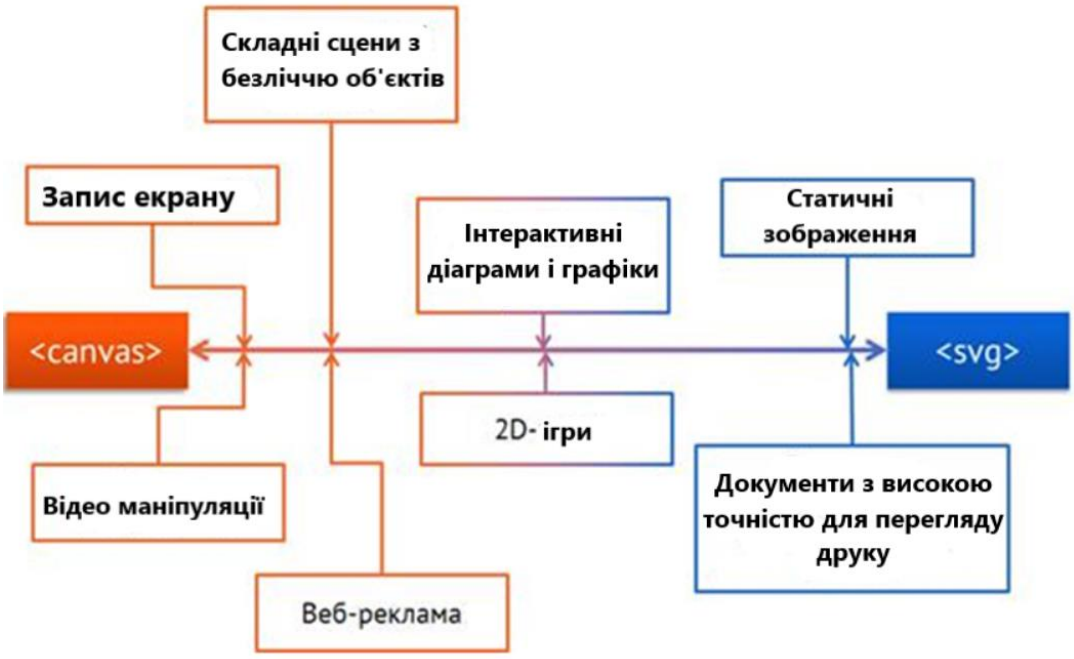

Рисунок 1.13. - Вибір залежить від поставленої задачі.

Переваги Canvas:

1. Зрозумілість. Canvas нескладний. У разі якщо ви розумієте JavaScript і мали навик роботи з графікою крізь примітивні 2D-API, розібратися з Canvas для вас не складе труднощів. API Canvas містить в межах 40 і 20 способів атрибутів.

2. Швидкість. За допомогою апаратного прискорення. Canvas спроектований так, щоб бути швидким, але якісь речі на 1-ий погляд здаються непрактичними.

3. Інтеграція. Canvas - це HTML5 елемент і його можливо вставити в будь-куди. Canvas можливо поєднувати з відео або зображеннями.

Переваги SVG:

1. Допомога в браузерах. Виробники браузерів не мають всі шанси уникати що прецедент, власне що їм треба продавати HTML5, в разі якщо вони бажають залишатися на плаву в браузерному ринку. В наслідок цього зараз ми бачимо базисну допомогу SVG у всіх провідних браузерах, при цьому просторами і за допомогою апаратного прискорення при відображенні. Але маркетингу, природно, було б не досить для такого, щоб хапатися за SVG, в разі якби не 3-ий момент.

2. SVG розвивається. Насправді, в разі якщо ще кілька років тому SVG розглядався перевагу в якості еталону для опису графіки, то зараз на нього дивляться вже важливо ширше.

Власне що стосується продуктивності, то тут все елементарно: до конкретного значення залежності швидкості програвання Canvas і SVG від обсягу екрану і числа об'єктів рухаються поблизу, але далі як виявилося, власне що SVG ніж будь-якого іншого підходить для зображень величезного обсягу, а Canvas демонструє себе спритним при величезний числа об'єктів.

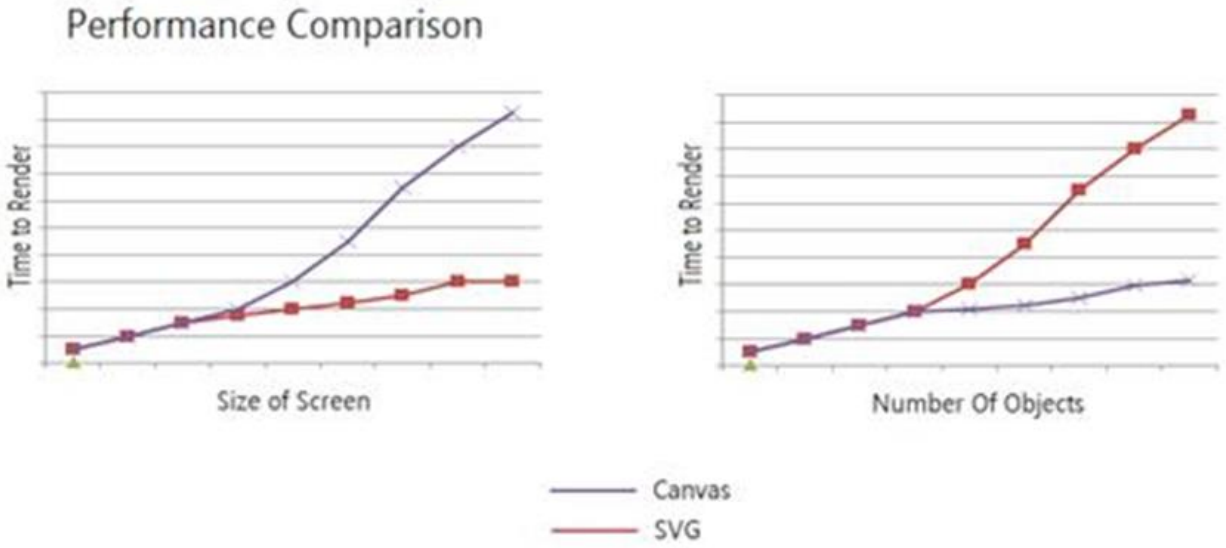

Рисунок 1.14. - Продуктивність Canvas vs SVG.

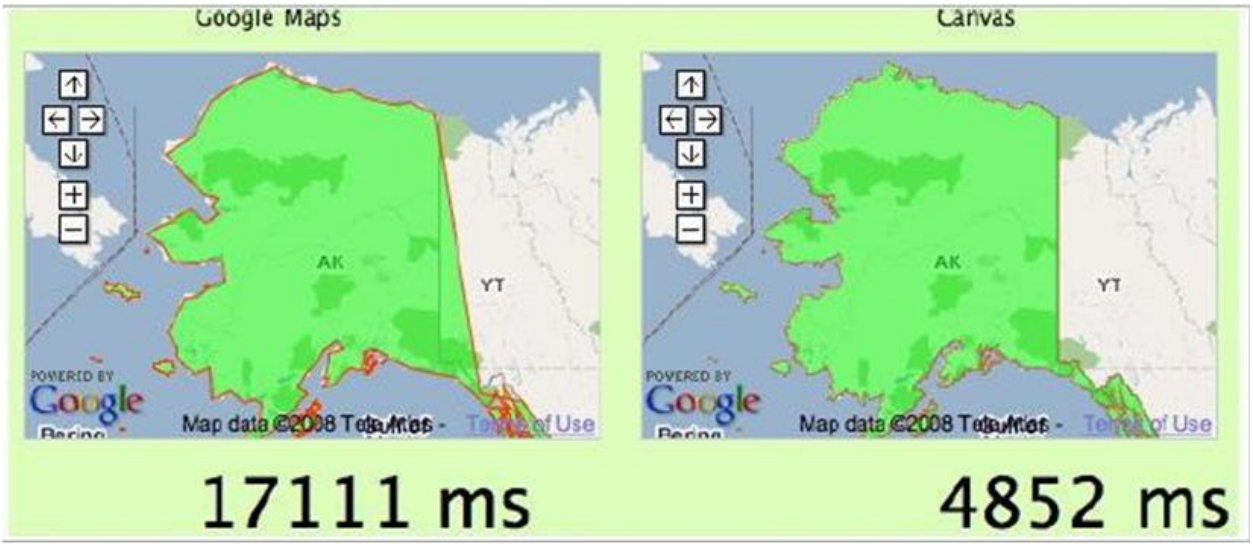

Рисунок 1.16. – Швидкодія на зображенні в Google Maps

Таблиця 1.2. - Переваги Canvas та SVG.

Canvas	**SVG**
Заснована на пікселях (динамічний png-файл)	Заснована на фігурах
Один елемент HTML	Кілька графічних елементів, які стають частиною моделі DOM
Змінюється тільки за допомогою сценаріїв	Змінюється за допомогою сценаріїв і CSS
Модель подій і взаємодія з користувачем засновані на пікселях (x, y)	Модель подій і взаємодія з користувачем є абстрактними (прямокутник, контур)
Продуктивність вище на невеликих поверхнях, при великому числі об'єктів (> 10 000) або для обох умов	Продуктивність вище на великих поверхнях, при невеликому числі об'єктів (<10 000) або для обох умов

Таблиця 1.3. – Порівняння найпопулярніших платформ.

Критерії	Мультимедійні платформи		
	HTML5 Canvas	SVG	Flash
Де працює	Елементи Canvas працюють в будь-якому веб-браузері HTML5. Незважаючи на те, що вони не працюють в IE до версії 9, вони працюють на ОС IOS і Android смартфонів і планшетів.	Працюють в більшості сучасних браузерів, IE не підтримує його до версії 9. Він також працює на ОС IOS і Android смартфонів і планшетів.	Flash працює в будь-якому браузері, який має плагін для нього. Проте, Safari для Mac 10.7 (Lion) НЕ не поставляється з плагіном Flash і Adobe більше не підтримує флеш на мобільних пристроях.
як створити	Більшість створення CANVAS здійснюється вручну, але деякі програми ілюстрації отримують підтримку HTML5, як правило, за допомогою плагінів.	Більшість векторних графічних програм може виводити файли як SVG.	Є багато комерційних додатків, доступних автору флеш-анімації.
Тип графіки	Елемент CANVAS створює растрові дані на екрані, але він може малювати вектори. Більшість з них створені вручну	SVG - це інструмент векторної графіки, але він може використовувати растрові файли,	Flash повністю векторний, але можна добавити растрові зображення в ваших флеш-файли.

	на JavaScript.	такі як JPG або PNG.	
вбудований HTML	Елемент CANVAS є елементом HTML5 і так автоматично використовується в HTML.	SVG має деяку підтримку браузера для вбудованих елементів в HTML5 (IE 9, FF12, Chrome 19 і Safari 5.1), але і в цілому XML вбудований.	Flash-файли SWF вкладається як файли і мають використовувати плагін для браузера, щоб працювати.
Необхідні технології	Елемент CANVAS використовує JavaScript для перегляду зображень. Якщо JavaScript відключений, CANVAS буде порожнім.	SVG вимагає сучасні браузери, які підтримують XML або SVG рендеринг.	Flash вимагає плагін для відображення зображень.
Швидкодія	CANVAS дуже швидкий для рендеринга зображень, але це вимагає аналізу JavaScript.	SVG, коли використовується в HTML5, дуже швидкий, тому що він не вимагає додаткових завантажень.	Flash є найповільнішим, що вимагає час завантаження як SWF-файлу і вкладення.
Пошукова оптимізація	CANVAS є основаним на тексті,	SVG є основаним на тексті, і може	Flash впроваджений SWF-файл, який

	і може бути прочитаний пошуковими системами.	бути прочитаний пошуковими системами.	може бути важким для пошукових систем, щоб розібрати.
Середній розмір готового проекту	1.7 Мб	1.5 Мб	> 2.5 Мб
Що може бути інтерактивним	Немає окремих елементів, що є клікабельними в CANVAS, тільки весь елемент.	SVG дозволяє клікати по елементам в зображенні.	Flash може мати інтерактивні точки, створені в будь-якому місці файлу.

Таблиця 1.4 Кількість fps у різних браузерах та технологіях.

Браузер	HTML	Canvas	SVG	Flash
Chrome	21	30	18	44
Firefox	9	38	2	44
IE	16	34	25	50
Opera	29	28	12	50
Safari	23	31	51	50

Висновки до розділу

У розділі розглянуто історію та теоретичні основи сучасних браузері. Приведено базові характеристики та статистику використання у різних країнах світу. Проведено огляд технологій створення графіки. Проаналізовано та неведено порівняння векторної та растрової графіки на різних платформах у використанні сучасних браузерах.

2. ПРОГРАМНІ ЗАСОБИ СТВОРЕННЯ ГРАФІЧНИХ МОДЕЛЕЙ.

2.1. Blender та Blend4Web.

Blender - професійний пакет для створення тривимірної графіки, що розвивається некомерційною організацією Blender Foundation. Одна з цілей організації - надати мережевого співтовариства з усього світу доступ до тривимірним технологіям в широкому сенсі, з Blender як центрального вузла системи. Blender підходить для використання як невеликими студіями, так і великими компаніями. В даний час Blender - один з найбільш популярних пакетів для роботи з тривимірною графікою. Також це безкоштовне програмне забезпечення, розповсюджується безкоштовно, доступні версії як для Windows, так і для Linux і macOS. Інтерфейс програми переведений на всі основні мови світу і може бути налаштований відповідно до вимог користувача.

Blender - популярна і добре відома програма. Можливості Blender добре задокументовані, як офіційно, так і користувачами.

На сьогоднішній день Blender є готовим рішенням для повного циклу роботи з тривимірною графікою. Включає не тільки засоби моделювання, анімації, візуалізації і обробки поста, але і інструменти монтажу, композинга і роботи зі звуком. Для розширення функціональності використовуються аддони. Для роботи не потрібно ніяких додаткових програм, модель або ціла сцена може бути повністю створена в Blender.

Характерною особливістю пакету Blender є його невеликий розмір в порівнянні з іншими популярними пакетами для 3D-моделювання. Демонстраційні сцени можна скачати на офіційному сайті або на сайті відкритих проектів «Blender Cloud».

2.1.1. Функції, інтерфейс, та особливості використання.

Функції пакета:

- Підтримка геометричних примітивів, охоплюючи полігональні моделі, систему швидкого моделювання, криві Безьє, площині NURBS, metaballs (метасфери), скульптурне моделювання та векторні шрифти.
- Універсальні інтегровані механізми рендеринга і інтеграція з зовнішніми рендерер YafRay, LuxRender і майже всіма іншими.
- Інструменти анімації, серед яких інверсна кінематика, скелетна мультиплікація і сіткова деструкція, головні кадри, нелінійна мультиплікація, редагування вагових коефіцієнтів вершин, обмежувачі.
- Динаміка піддатливих тіл (включаючи визначення колізій об'єктів при взаємодії), динаміка твердих тіл на базі фізіологічного движка Bullet.
- Система частинок включає в себе систему волосся на базі частинок.
- Модифікатори для використання неруйнівних ефектів.
- Мова програмування Python застосовується як засіб визначення інтерфейсу.
- Базисні функції нелінійного відео та аудіо монтажу.
- Композинг відео, робота з хромакея.
- Відстеження відеокамери і об'єктів.
- Real-time контроль при фізіологічній симуляції і рендеринга.
- Процедурне і node-based текстурирование, а ще ймовірність зображати текстуру навпростець на моделі.
- Grease Pencil - інструмент для 2D анімації в повному 3D Пайплайн.
- Blender Game Engine.

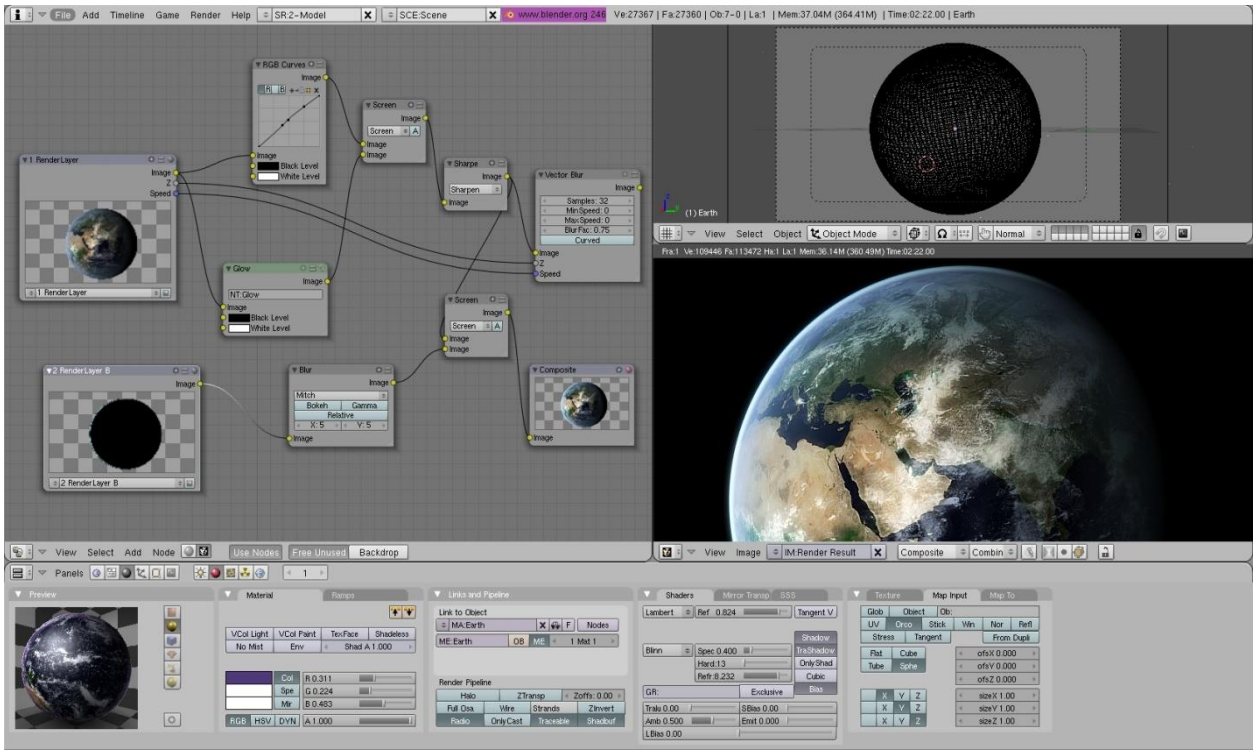

Рисунок 2.1. Знімок екрана інтерфейсу Blender 2.4.

Blender мав репутацію програми, важкою для дослідження. Буквально будь-яка функція містить хитросплетіння кнопок, і беручи до уваги чисельність ймовірностей, що надаються Blender, будь-яка кнопка інтегрована в більше ніж одне хитросплетіння (shortcut). З тих часів як Blender став програмою з не закритим початковим кодом, були додані повні контекстні раціони до всіх функцій, а впровадження інструментів створено більше логічним і гнучким. Подальше вдосконалення призначеного для користувача інтерфейсу з вступом колірних схем, прозорих плаваючих складових, свіжої системою перегляду дерева об'єктів і різними невеликими змінами.

Відмінні риси інтерфейсу користувача:

Режими редагування. Два провідних режиму: Об'єктний режим (Object mode) і Режим редагування (Edit mode), які перемикаються кнопкою Tab. Об'єктний режим в основному застосовується для маніпуляцій з персональними об'єктами, в той час як режим редагування - для маніпуляцій з фактичними даним.

Широке використання гарячих кнопок. Основна маса команд проводиться з підтримкою клавіатури. До виникнення 2.x і тим більше 2.3x-версії, це була єдина можливість виконувати команди.

Управління робочим простором. Графічний інтерфейс Blender зроблено з 1-го або ж декількох екранів, кожен з яких має можливість бути поділений на секції і підсекції, які мають всі шанси бути всякий частиною інтерфейсу Blender'a. Графічні складові будь-якої секції можуть контролюватися тим самим інструментом, що і в 3D-просторі, наприклад можливо скорочувати і нарощувати кнопки інструментів що ж методом, власне що і в 3D-перегляді. Юзер цілком тримає під контролем розташування і компанію графічного інтерфейсу. Дана манера графічного інтерфейсу досить подібна на манеру, застосовувану в редакторі карт UnrealEd для гри Unreal Tournament.

Робочий простір Blender є одним з найбільш новаторських концепцій графічного інтерфейсу для графічних інструментів і натхненним дизайном графічного інтерфейсу запатентованих програм, таких як Luxology's Modo.

Допоміжні особливості:

Внутрішня файлова система, що дозволяє берегти декілька сцен в єдиному файлі (званому .blend-файл).

Всі «.blend»-файли сумісні як з більше старими, наприклад і з більше свіжими версіями Blender.

Blender готує запасні копії планів на всій роботі програми, власне що дозволяє зберегти дані при несподіваних життєвих обставинах.

Всі сцени, об'єкти, матеріали, текстури, звуки, зображення, post-production-ефекти можуть бути збережені в єдиний файл формату .blend.

Налаштування робочого простору можуть бути збережені в .blend-файл, завдяки чому при завантаженні файлу користувач отримає саме те, власне що зберіг в нього.

Експорт в Web:

- Blend4Web - WebGL-фреймворк дозволяє експортувати підготовлені в Blender сцени для відтворення в стандартних браузерах, без необхідності установки будь-яких розширень.
- Verge3D - Рендер в реальному часі і інструментарій для створення інтерактивних 3D веб-інтерфейсів, працює поверх Blender і 3ds Max.
- Armory - 3D ігровий движок з відкритим вихідним кодом з повною інтеграцією в Blender, може створювати WebGL програми та ігри.

2.1.2. Blend4Web та інтеграція з Blender.

Blend4Web - відкритий фреймворк, призначений для створення і відображення інтерактивної тривимірної графіки в браузерах. Blend4Web використовує відкритий пакет 3D-моделювання Blender для підготовки контенту. Розроблено та випущено в 2014 році російською фірмою «Тріумф».

На початку розробки Blend4Web замислювався як інструмент для створення WebGL контенту в Blender. В даний час Blend4Web є повнофункціональним інтегрованим рішенням, ключовим компонентом якого залишається Blender. Компанія активно співпрацюємо з проектом Blender. Зокрема, компанія є діамантовим спонсором проекту. Крім того, розробники беруть активну участь у розвитку Blender.

Підготовка аудіо-візуального контенту для Blend4Web здійснюється в пакеті 3D-моделювання та анімації Blender. Контент відтворюється за допомогою WebGL, Web Audio і інших стандартних браузерних технологій, без використання будь-яких розширень.

Фреймворк поширюється на умовах подвійного ліцензування, під відкритою GPLv3 і комерційної ліцензіями. Вихідний код знаходиться в репозиторії на GitHub.

3D-сцена створюється в Blender і експортується у вигляді файлу в форматі JSON і виконуваного файлу для подальшого завантаження веб-додатком. Сцена

може бути також експортована у вигляді єдиного самодостатнього HTML-файлу, в який упаковуються експортовані дані, призначений для користувача інтерфейс веб-плеєра і виконуваний код. Експорт в HTML-форматі розглядається як найбільш простий спосіб роботи з Blend4Web. Підсумковий файл з мінімальним розміром близько 1 МБ може бути розміщений на веб-сторінці за допомогою стандартного елемента <iframe>.

До складу програмних компонентів Blend4Web входять бібліотеки JavaScript, доповнення до Blender і набір інструментів для налаштування параметрів 3D-сцен, налагодження та оптимізації.

Особливості:

Фреймворк містить ряд компонентів, зазвичай присутніх в ігрових движках, включаючи систему позиціонування джерел звуку, фізичний движок, систему анімації і шар абстракції для програмування ігрової логіки.

На кожен об'єкт може бути призначено до 8 анімацій різного типу, включаючи кісткову і верхову анімацію. Швидкість і напрямок анімації, а також параметри систем частинок (розмір, початкова швидкість і кількість) можуть бути задані за допомогою API.

Підтримуються динамічне завантаження і вивантаження даних, карти освітлення, є можливість симуляції підповерхневого розсіювання.

Є рішення для рендеринга зовнішніх сцен, включаючи симуляцію впливу вітру, симуляцію води, атмосферні ефекти.

З огляду на використання кросбраузерності технології WebGL, Blend4Web працює у всіх основних сучасних браузерах, в тому числі на мобільних пристроях. Проте, в браузерах з експериментальної реалізацією стандарту WebGL, таких як Internet Explorer, можливості Blend4Web підтримуються в повному обсязі. Існують додатки, здатні працювати в операційній системі Tizen, зокрема, на «розумних» годинниках Samsung.

Серед нетривіальних для браузерних фреймворків особливостей називаються такі: оптимізація методом комбінування викликів відтворення,

оптимізація методом відсікання прихованих об'єктів, винесення фізичних розрахунків в окремий потік виконання, симуляція поведінки морських хвиль.

У версії 14.09 в Blend4Web реалізована можливість інтерактивності в 3D-сценах за допомогою інструменту для візуального програмування. Інструмент нагадує редактор логіки вбудованого ігрового движка Blender, зокрема, в ньому, як і в BGE, використовуються вибудовуються в інтерфейсі Blender візуальні логічні блоки. Анімаційні послідовності, створені художником, програються у відповідь на взаємодію користувача з тими чи іншими тривимірними об'єктами.

Починаючи з версії 15.03 Blend4Web підтримує приєднання двовимірних HTML-елементів до 3D-об'єктів, а також копіювання об'єктів під час роботи програми.

Серед ефектів на основі постобробки підтримуються сяючі поверхні, засвітка яскравим світлом, глибина різкості камери, сутінкові промені, розмиття при русі і взаємне затінення.

Підтримка пристроїв віртуальної реальності була реалізована в наприкінці 2015р. Робота шолома Oculus Rift забезпечується експериментальним API WebVR.

Інтеграція з Blender:

Доповнення для Blender написано на Python і Ci, і має можливість бути скомпільовано для платформ Linux, OS X та Windows.

Профіль налаштувань Blend4Web активується в налаштуваннях аддона. При перемиканні в профіль відбувається перестроювання інтерфейсу Blender з одночасним видаленням всіх підтримуються налаштувань.

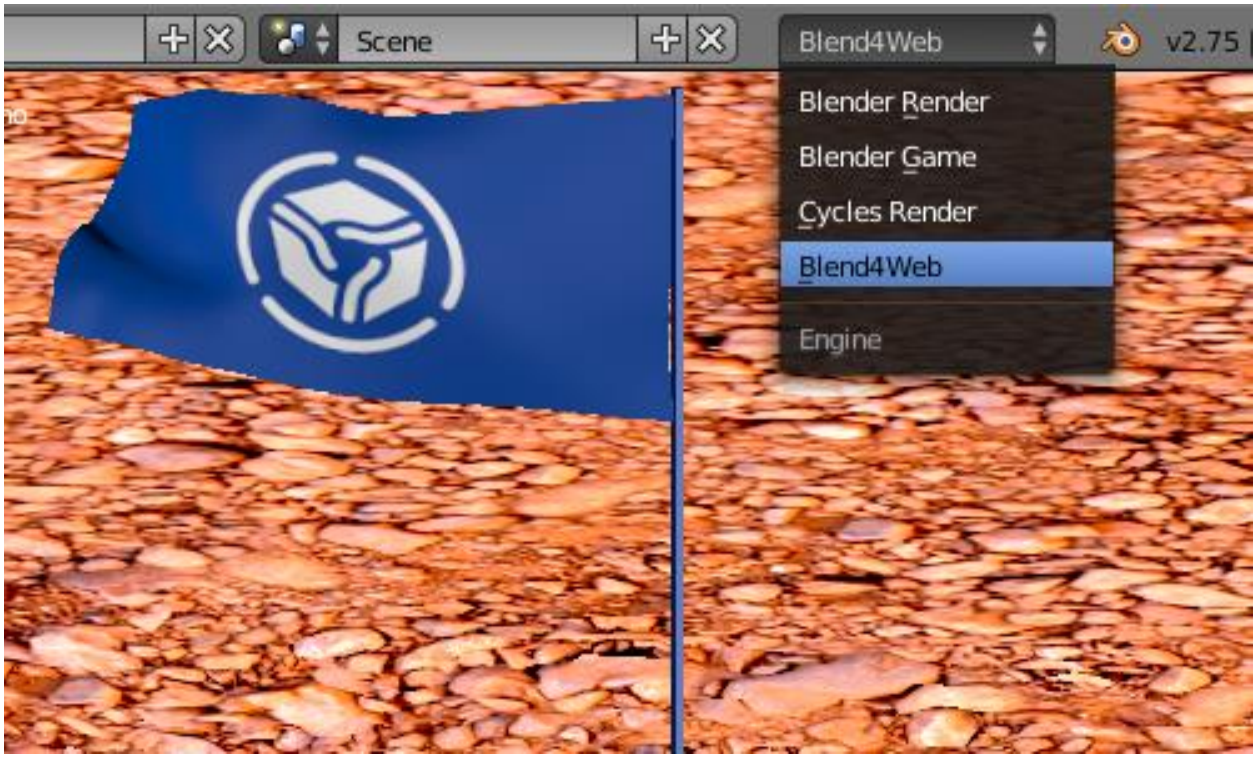

Рисунок 2.2. Меню перемикання в профіль Blend4Web.

Blend4Web підтримує ряд специфічних для Blender функцій, таких як вузловий редактор матеріалів (інструмент для візуального програмування шейдеров) і система частинок. Також є початкова підтримка редактора нелінійної анімації (NLA) в складі пакету Blender, призначеного для створення простих сценаріїв.

Порівняння з аналогами:

- Blender Game Engine

Як і вбудований ігровий движок Blender (Blender Game Engine, BGE), Blend4Web завантажує підготовлені в Blender дані сцен без використання будь-якого проміжного редактора. Додатки на основі Blend4Web призначені для роботи в браузерах, в той час як виконувані файли BGE запускаються поза браузерів. API Blend4Web реалізовані на JavaScript, на відміну від заснованих на Python API вбудованого движка Blender. Замість візуального редактора логіки BGE, непідтримуваного в Blend4Web, в останньому передбачається використання стандартних підходів, прийнятих в веб-розробці.

- Unity

У Unity можливість експорту в WebGL є з версії 5. Реалізація підтримки WebGL, проте, носить експериментальний характер, зокрема, в WebGL-версії в найближчій перспективі не планується підтримка мобільних пристроїв, системи позиціонування джерел звуку, відеотекстур.

- Three.js

WebGL-бібліотека Three.js вимагає написання програмного коду для виконання тривіальних завдань, що підвищує поріг входження і подовжує навчання; в Blend4Web велика частина робіт з підготовки онлайн-презентацій не вимагає програмування.

Відомі випадки використання:

НАСА розробило інтерактивну веб-додаток до третьої річниці від дня посадки марсохода К'юріосіті. У додатку, створеному на основі Blend4Web, реалізовано рух ровера, управління камерами і маніпулятором, а також відтворені деякі відомі події місії. Додаток було представлено на початку секції WebGL на конференції SIGGRAPH 2015.

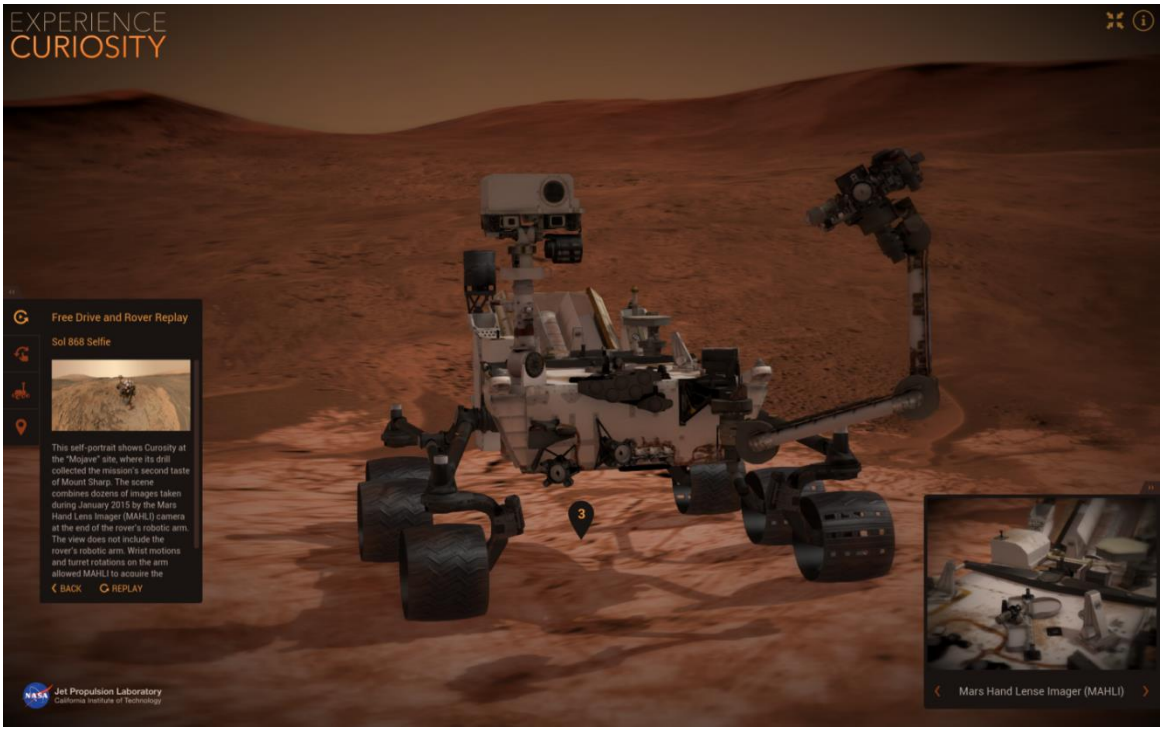

Рисунок 2.3. Experience Curiosity: ровер робить Селфі.

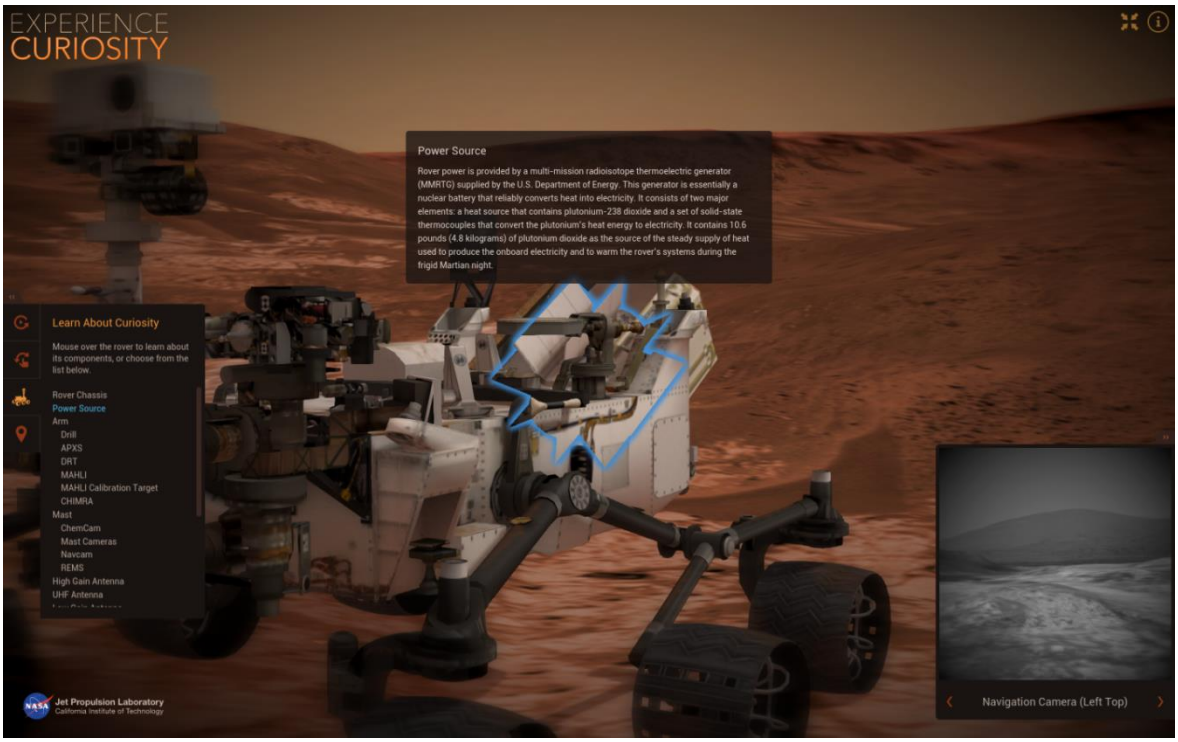

Рисунок 2.4. Experience Curiosity: навчальний режим.

Рисунок 2.5. Experience Curiosity: режим вільного управління.

2.2. 3ds Max

Autodesk 3ds Max - професійне програмне забезпечення для 3D-моделювання, анімації і візуалізації при створенні ігор і проектуванні. В даний час розробляється і видається компанією Autodesk. Використовується у виробництві фільмів. Ось тільки короткий список фільмів, створених із застосуванням 3D Max: «Гаррі Поттер і в'язень Азкабану», «Лара Крофт: розкрадачка гробниць», «Людина-павук», «Люди Х», «Історія іграшок». «Трансформери 2» і сотні, сотні інших. Про рекламні ролики і кліпи навіть не йде мова - їх сотні тисяч.

2.2.1. Методи моделювання.

Історія пакета

Перша версія пакету була випущена в 1990 році. За цей час розробками пакету промишляла автономна майстерня Yost Group, розроблена розробником програмного забезпечення Гарі Йост. Autodesk на перших порах промишляв лише тільки виданням пакета.

Перші чотири мали назву 3D Studio DOS (1990-1994 роки). Потім пакет був перейменований в 3D Studio MAX (1996-1999 роки). У 2000-2004 роках пакет випускався під маркою Discreet 3dsmax, а з 2005 року - Autodesk 3ds Max.

Моделювання.

3ds Max має гігантськими способами для створення всіляких за формою і труднощі тривимірних комп'ютерних моделей, справжніх або ж чудових об'єктів навколишнього світу з впровадженням різноманітних технік і пристроїв, що включають:

- Полігональне моделювання, Editable mesh і Editable poly - це найвідоміший спосіб моделювання, застосовується для створення важких моделей і низькополігональних моделей для ігор. Як правило, моделювання важких об'єктів з наступним перетворенням в Editable

poly настає з зведення параметричного об'єкта «Box», і в наслідок цього метод моделювання прийнято іменувати «Box modeling».

- Моделювання на базі неоднорідних оптимальних B-сплайнів (NURBS).
- Моделювання на базі «Сіток шматків» або ж площин Безьє - підходить для моделювання тіл обертання.
- Моделювання на базі впровадженням інтегрованих бібліотек нормальних параметричних об'єктів (примітивів) і модифікаторів.
- Моделювання на базі сплайнів (Spline) з наступним використанням модифікатора Surface - простий аналог NURBS, сприятливий, втім, для створення об'єктів зі складними перетікають формами, які непросто зробити способами полігонального моделювання.
- Моделювання на базі сплайнів з наступним використанням модифікаторів Extrude, Lathe, Bevel Profile або ж створення на базі сплайнів об'єктів Loft. Даний метод широко використовується для будівельного моделювання.

Способи моделювання можуть поєднуватися один з одним.

Моделювання на базі нормальних об'єктів, як правило, вважається головним способом моделювання і працює вихідною точкою для створення об'єктів важкою структури, власне що пов'язано з впровадженням примітивів в поєднанні один з одним як примітивних частинок складових об'єктів.

Звичайний об'єкт «Чайник» (Teapot) заходить в даний комплект в міць історичних підстав: він застосовується для досліджень матеріалів і освітлення в сцені, і, беручи до уваги такого, давним-давно став специфічною емблемою тривимірної графіки.

Particle Systems

Particle Systems - це сукупність малорозмірних об'єктів, керованих цілісного ряду характеристик. Прикладами ситуацій, в яких трапляються системи частинок, можуть служити сцени, де треба створити зливу, снігопад, дим, полум'я, зоряне

небо, потоку фонтану, іскри. Починаючи з 8 версії є 7 провідних джерел частинок, показують різну поведінку.

- PF Source (Джерело Particle Flow) - потік частинок, здатних реагувати на запрограмовані у вбудованій системі Particle Flow події. Такий потік частинок може імітувати що завгодно - від бризок фонтану до димового шлейфу реактивного двигуна ракети.
- Spray (Бризки) - створює спрощений варіант ефекту бризок, на зразок крапель дощу, і має кілька параметрів для налаштування форми частинок, їх розміру та характеру падіння.
- Super Spray (Супер бризки) - істотно вдосконалена в порівнянні зі стандартною система бризок, що має безліч параметрів для визначення характеру народження, руху і форми частинок. Частинкам можна надавати форму різних об'єктів або дозволяти з'єднуватися між собою на зразок водяних крапель.
- Snow (Сніг) - створює простий ефект падаючого снігу і має багато параметрів для налаштування форми частинок, їх розміру та характеру падіння.
- Blizzard (Заметіль) - істотно вдосконалена версія частинок Snow (Сніг). Частинкам можна надавати форму різних об'єктів;
- PArray або Particle Array (Масив частинок) - підходить для моделювання частинок будь-якого типу, а також для вдосконалених ефектів імітації вибуху. Частинкам можна надавати форму різних об'єктів.
- PCloud або Particle Cloud (Хмара частинок) - створює статичне хмара частинок і може застосовуватися для імітації тривимірних зоряних полів, косяка риб або зграї птахів. Частинкам можна надавати форму різних об'єктів.

Reactor

3ds Max включав пристрій розрахунку фізики Reactor, створений фірмою Havok. Reactor дозволяє моделювати поведінку твердих тіл, піддатливих тіл,

тканини з урахуванням сили тяжіння та інших впливів. Починаючи з версії 2012 Reactor виключений з пакету. На заміну йому прийшов модуль MassFX.

2.2.2. Візуалізація та модулі

Візуалізація

Візуалізація є заключним етапом роботи над сценою. В переважній більшості випадків робота зі сценою створюється в спрощеному вигляді: розмір текстур маленький, тіні і джерела світла, різні властивості матеріалів відключені, складна геометрія і різні ефекти не відображаються. Тільки після візуалізації стає видно всі властивості матеріалів об'єктів і проявляються ефекти зовнішнього середовища, застосовані в складі сцени. Для виведення кінцевого зображення на екран вибирають необхідний модуль візуалізації (МВ), який за допомогою математичних алгоритмів зробить обчислення зовнішнього вигляду сцени з усіма необхідними ефектами. При цьому, час розрахунку може варіюватися від частки секунди до декількох місяців, в залежності від складності завдання. Більшість МВ є окремими програмами, що вбудовуються як доповнення в 3ds Max. Для досягнення найкращого результату необхідно, щоб одиниці виміру сцени були виставлені правильно. Якщо модель має розміри, відповідні реальності, то і освітлення буде найбільш реалістичним.

Список модулів візуалізації:

- Scanline

Визуалізатор за замовчуванням в 3ds Max. Вихідним методом візуалізації в 3DS Max є скануючий порядковий алгоритм. Деякі розширені можливості були додані в Scanline через роки, такі як розрахунок Global Illumination, Ray Tracing і Radiosity, проте більшість функцій перейшло до нього від інших визуалізаторів.

- Art Render

Вбудоване в 3ds max засіб рендеринга Autodesk Raytracer (ART) - це швидкий рендерер, заснований на процесорі, що ідеально підходить для рендеринга і анімації архітектурного, товарного та промислового дизайну. ART

дозволяє виконувати великі складні сцени і використовувати необмежений рендеринг на декількох комп'ютерах за допомогою Backburner.

- Arnold Renderer

Програмне забезпечення починаючи з версії 3ds Max 2018 представляє механізм рендеринга Arnold 5.0 як один з механізмів рендеринга за замовчуванням, замінюючи движок mental ray. Створений компанією Solid Angle. Має свої матеріали і освітлення.

- Mental ray

Mental ray є придатною для виробничого застосування високоякісної системою візуалізації, розробленої компанією Mental Images. Mental ray вбудований в останні версії 3DS Max, це потужний інструмент візуалізації, що підтримує сегментну візуалізацію (подібно механізму супроводжує візуалізації, реалізованому в Maya), а також технологію розподіленої візуалізації, що дозволяє раціонально розділяти обчислювальне навантаження між декількома комп'ютерами. Включається в 3ds Max версія Mental ray поставляється з набором інструментарію, що дозволяє відносно просто створювати безліч різних ефектів.

- V-Ray

Високоякісний фотореалістичний визуалізатор, спроектований в якості плагіна для 3ds Max. Дуже часто використовується професіоналами, часто замінюючи стандартний Scanline і Mental ray. Сумісний з більш старими версіями 3ds Max. Має власні матеріали, камери, джерела освітлення та атмосферні ефекти. Також в нього вбудована «система денного світла»: V-Ray Physical Camera, V-Ray Sky і V-Ray Sun (фізична камера, небо і сонце), використання яких в сукупності дозволяє отримати хороші результати навіть при стандартних налаштуваннях.

- RenderMan

Сторонній засіб підключення до конвеєра RenderMan, також корисно в тих випадках, коли потрібно інтеграція 3DS Max з системою візуалізації Renderman. Конект з 3DS Max відбувається за допомогою DoberMan.

- FinalRender

Зовнішній візуалізатор компанії Cebas. Є найбільш повним фотон-заснованим візуалізатором, поступаючись своїми можливостями лише Mental ray. Перевага полягає в щільній інтеграції з іншими рішеннями Cebas, що забезпечують широкий спектр різноманітних атмосферних, лінзових ефектів та ін., Чого немає у інших визуализаторов.

- Brazil R / S

Високоякісна, фотореалістична система візуалізації зображення, розроблена компанією SplutterFish Llc. У цьому візуалізаторі присутні кілька алгоритмів прорахунку глобального освітлення Global Illumination: QMC і Photon Mapping. Brazil добре зарекомендував себе серед архітекторів, дизайнерів і художників комп'ютерної графіки, завдяки простоті налаштувань, стабільності і якісного результату візуалізації.

- Fryrender

Фотореалістичний, заснований на законах фізики, спектральний визуализатор. Створено компанією RandomControl. Надає можливість отримувати зображення найвищої якості і досягати природного реалізму.

- Indigo Renderer

Фізично коректний рендер. Основна особливість його в тому, що всі розрахунки світла, енергії, і т. д. Відбуваються взаємозалежно, що і відрізняє його від інших рендерів, де все роздільно і визначається самим користувачем.

- Maxwell Render

Є першою системою візуалізації, в якій прийнята «фізична парадигма». В основу всієї системи покладені математичні рівняння, що описують поведінку світла. Вводячи в обіг реальні фізичні закони, Maxwell Render дозволяє уникнути тривалого і тонкого процесу настройки параметрів візуалізації, який має місце в разі більшості визуализаторов, які працюють за іншими алгоритмами.

- LuxRender

Система на фізичному рівні коректної візуалізації тривимірних сцен, містить щирий відправною код. Для застосування системи LuxRender, потрібно

вивозити сцени і моделі з редакторів графіки з підтримкою особливих плагінів або ж скриптів.

- Kerkythea

Система візуалізації, що дозволяє створювати фотореалістичні зображення. Використовує фізично точні матеріали і освітлення. Kerkythea 2008 Echo має свій власний редактор сцени і матеріалів, володіє простим і зручним інтерфейсом.

- Verge3D

3D рендерер реального часу і супутній інструментарій, призначений для створення і відображення інтерактивної тривимірної графіки в браузерах.

Модулі

3ds Max володіє досить великою базою стандартних засобів, що полегшують моделювання всіляких спецефектів. Крім стандартної бази існує маса додаткових коштів (плагінів) дозволяють не тільки створювати значно більш реалістичні ефекти вогню, води, диму, але містять додаткові інструменти моделювання. Модулі є зовнішніми вбудовуваними модулями, які продаються окремо від пакета 3ds Max або ж поширюються безкоштовно через Інтернет. Дані програми створюються як великими компаніями, що спеціалізуються з розробки програмного забезпечення, так і простими розробниками-ентузіастами. Додаткових модулів для 3ds Max настільки багато, що кількість інструментів пропонованих ними у багато разів перевершує комплект стандартних засобів 3ds Max. Модулі спрощують виконання багатьох завдань - наприклад, дозволяють витрачати менше часу на прорахунок візуалізації (за рахунок більш вдосконалених підключаються визуализаторов) або прискорюють моделювання об'єктів, завдяки різноманітним модифікаторам і більше функцій. Такі додаткові модулі як Particle Flow, Cloth FX, Reactor, - стали настільки популярні, що було вирішено інтегрувати їх в програму 3ds Max і тепер вони є частиною програми. Нижче представлений список деяких плагінів для 3ds Max:

- FumeFX - фотореалістичні ефекти вогню, мов полум'я, диму.
- Phoenix FD - аналог FumeFX від Chaos Group для створення вогню і диму.

- DreamScape - реалістичні ландшафти, гори, небо, атмосферні ефекти і т. Д.
- AfterBurn - фотореалістичні ефекти хмар, диму, вибуху і т. Д.
- GrowFX - рослини будь-якого виду: від пальм і ліан до сосен, від квітів до великих широколистяних дерев і т. д. Кожна рослина створене за допомогою цього плагіна можна вільно анімувати.

2.2.3. Verge3D

Verge3D - це рендеринг у реальному часі та інструментарій, який дозволяє створювати багатий інтерактивний досвід роботи на веб-сайтах. Завдяки базовій технології WebGL тривимірний вміст можна переглядати за допомогою звичайного веб-браузера, в тому числі на мобільних пристроях, і не потребує жодного.

Найхарактернішою властивістю робочого процесу Verge3D є те, що графічний вміст є автором всередині моделюючого середовища, з якого він перетворюється на готовий до Інтернету додаток для розгортання в Інтернеті чи офлайн.

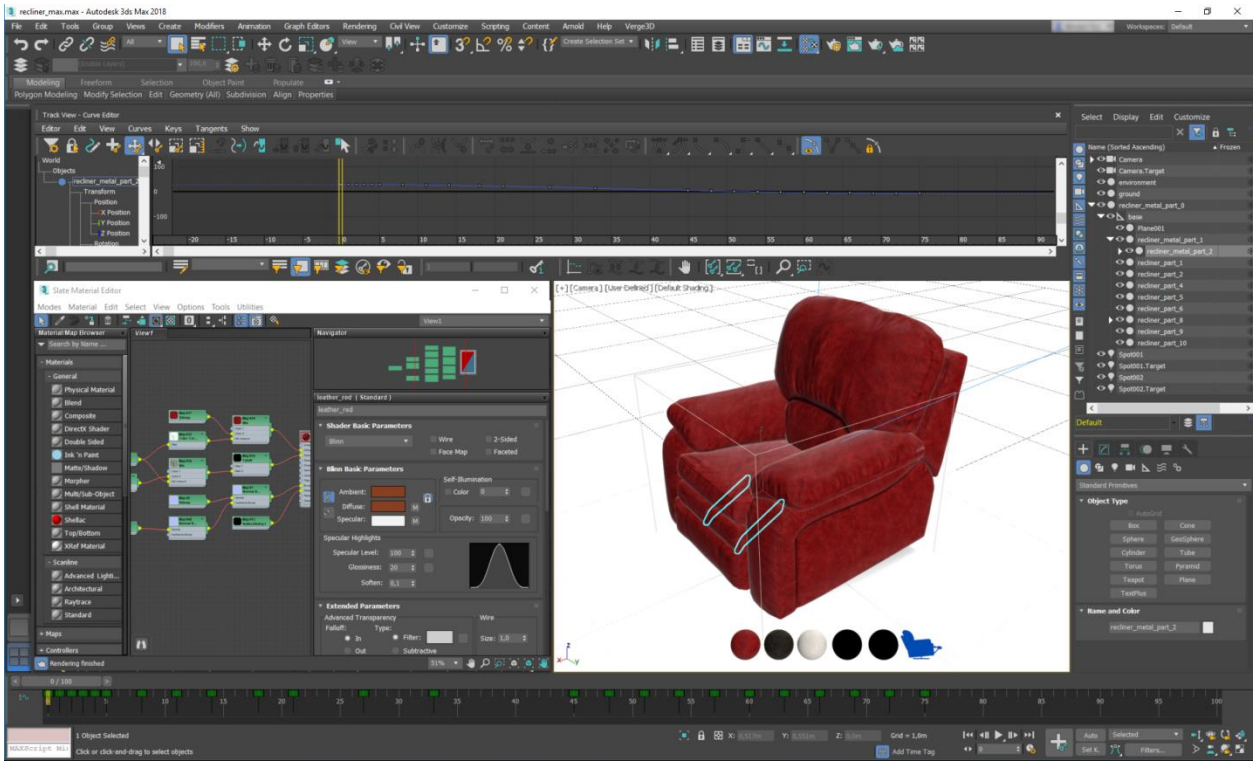

Рисунок 2.6. Інтерефейс робочого середовища.

Окрім Autodesk 3ds Max, Verge3D пропонує також інтеграційний модуль для Blender, відомого набору з 3D-моделювання та анімації з відкритим кодом.

Основні риси

Користувачі 3ds Max можуть насолоджуватися використанням стандартних або PBR матеріалів будь-якої складності, текстури навколишнього середовища, освітлення, камер, груп, анімації тощо. Вміст можна переглянути прямо у вікні перегляду, створеній у режимі високої якості, або у веб-переглядачі за допомогою Sneak Функція попереднього перегляду та експортується для Verge3D у форматі glTF 2.0.

Інтерактивність можна легко додати за допомогою Пазлів, середовищ візуального сценарію, подібних до подряпин. Ви можете побудувати повномасштабну тривимірну веб-програму, будь то конфігуратор продукту, програма електронного навчання або програмний компонент корпоративного програмного забезпечення - все без написання єдиного рядка коду. З іншого боку, веб-розробники можуть використовувати величезний API на основі Three.js для інтеграції програм Verge3D у будь-яку існуючу або заплановану інфраструктуру.

Завершені тривимірні веб-додатки можуть бути як власноруч розміщені, так і розгорнуті в мережі Verge3D, хмарному сховищі та CDN, що працює на серверах Amazon.

Висновки до розділу

У розділі розглянуто програми для створення графічних моделей. Описано роботу програми Blender та 3dsMax та наведено базові функції, інтерфейс і особливості використання. Також проаналізовано методи інтеграції з графічними бібліотеками. Наведено порівняння з аналогами, які представлені компаніями розробниками самих програм для створення графічних моделей.

3. ОСОБЛИВОСТІ ЗАСТОСУВАННЯ WEBGL. ПОРІВНЯННЯ ГРАФІЧНИХ БІБЛІОТЕК.

3.1. WebGL

WebGL спочатку була заснована на OpenGL ES 2.0 (Embedded Systems), версії специфікації OpenGL для таких пристроїв як iPhone від Apple і iPad. Але специфікація розвивалася, стала незалежною, її основна мета це забезпечення переносимості між різними операційними системами і пристроями. Ідея веб-інтерфейсу, рендеринг в реальному часі відкрили новий всесвіт можливостей для веб-3D середовищ, таких як відеоігри, наукова та медична візуалізація. Крім того, через широке поширення веб-браузерів, ці та інші види 3D-додатків можуть бути запущені на мобільних пристроях, таких як смартфони і планшети.

Системні вимоги:

WebGL є веб-основою API 3D-графіки. Як така вона не вимагає установки. Відповідність браузерів:

- Firefox 4.0 +
- Google Chrome 11 +
- Safari (OSX 10.6 +)
- Opera 12 +

WebGL є 3D графічною бібліотекою, яка дозволяє сучасним інтернет-браузерам малювати 3D-сцени стандартним і ефективним способом. Рендеринг - це візуалізація процесу створення зображення з моделі за допомогою комп'ютерної програми. Оскільки цей процес виконується на комп'ютері, існують різні способи отримання таких зображень.

Перша відмінність, використання будь-яких спеціальних графічно-апаратних засобів чи ні. Програмний рендер, використовується в тих випадках, коли всі розрахунки, необхідні для відтворення 3D-сцен виконуються з використанням основного процесора комп'ютера; з іншого боку, використання апаратного рендеринга, є актуальним в тих випадках, коли є графічний процесор (GPU) для обчислення 3D-графіки в реальному часі. З технічної точки зору,

апаратний рендеринг є набагато більш ефективним в порівнянні з програмним, бо є спеціалізовані апаратні складові, які обробляють операції. Але, з іншого боку, програмний рендеринг зазвичай більш поширений через брак апаратних залежностей.

Друга відмінність, це процес рендеринга локально або віддалено. Коли зображення, яка має бути відображена складне, то рендеринг, швидше за все, буде відбуватися віддалено. Це випадок 3D-анімаційних фільмів, коли виділені сервери з великою кількістю апаратних ресурсів дозволяють рендерить складні сцени - це серверний рендеринг. Протилежністю цьому є рендеринг, що виконується локально - це клієнтський рендеринг.

WebGL має клієнто-орієнтований підхід; елементи, які складають частини 3D-сцени, зазвичай завантажуються з сервера. Однак, вся подальша обробка, необхідна для отримання зображення виконується локально, за допомогою графічного обладнання клієнта.

3.1.1. Структура WebGL додатку

Як і в будь-якій бібліотеці 3D-графіки, в WebGL необхідна присутність певних компонентів для створення 3D-сцен.

Елементи:

- Сцена - це 3D простір, в якому розташовуються потрібні об'єкти: елементи (меши) і джерела світла, які висвітлюють елементи з будь-якої сторони.
- Камера - точка, відображення простору сцени.
- Джерело світла - елемент, який створює освітлення. Він може бути розсіяним, точковим, спрямованим
- Меш - це елемент сцени, який складається з геометрії і матеріалу. 3D об'єкт.
- Геометрія - це набір вершин, які при генерації з'єднуються між собою графічними примітивами.

- Матеріал - це спосіб відображення і зовнішній вигляд елемента. Це те, як він себе веде в рамках представлення на сцені. Тіні і відображення - найпростіші приклади для демонстрації.
- Текстура - це зображення, яке може використовуватися в рамках матеріалу, щоб задати зовнішній вигляд об'єкта.

Принцип роботи WebGL:

1. Процес починається зі створення масиву вершин. Цей масив містить атрибути вершин: розташування в 3D просторі, інформацію про текстуру, колір або нормалей (освітлення). Дана інформація створюється в JavaScript з файлах опису 3D моделей (.obj файли) або з бібліотеки, яка описує масив вершин геометричних фігур.

2. Потім вершини відправляються на GPU. Разом з цим потрібно так само передати масив індексів вершин для контролю перетворення вершин в трикутники

3. GPU читає кожну вершину з буфера вершин і проганяє її через вершинні шейдери. Вершинні шейдери обчислюють положення вершини на екрані, колір, текстурні координати.

4. GPU з'єднує по 3 вершини в трикутники використовуючи масив індексів.

5. Потім відбувається растеризація, приведення зображення до формату піксельних фрагментів.

6. Сформовані пікселі проходять через піксельні шейдери. Піксельний шейдер розраховує колір і глибину кожного пікселя. Там же на цьому етапі відбувається накладання текстури і розрахунок освітлення. Розраховані пікселі поміщаються в буфер фреймів (кадрів).

7. Буфер кадрів - остання стадія відтворення. Результатом його роботи є 2D зображення на екрані з ефектами глибини.

3.1.2. WebGL API

WebGL-контекст

WebGL контекст с можливо отримати з DOM-елемента canvas, викликавши метод getContext.

```
var names = ["webgl", "experimental-webgl", "webkit-3d", "moz-webgl"];
    gl = null;
    for (var ii = 0; ii < names.length; ++ii) {
        try {
            gl = canvas.getContext(names[ii]);
        } catch(e) {}
        if (gl) {
            break;
        }
    }
```

Рисунок 3.1. Отримання WebGL контексту

При успішному отриманні контексту об'єкт gl має методи, назви яких дуже схожі на функції OpenGL ES. Функція clear (COLOR_BUFFER_BIT) для WebGL буде gl.clear (gl.COLOR_BUFFER_BIT), що дуже зручно. Але не всі функції WebGL мають такий же синтаксис, як і функції OpenGL ES 2.0.

Шейдери

Шейдерна програма є невід'ємною частиною побудови зображень за допомогою WebGL. Саме через неї задається положення і колір кожної вершини наших ліній. У нашій задачі використовується два шейдера: вершинний і фрагментний. При побудові ліній в тривимірному просторі вершинний шейдер відповідає за стан вершин в просторі, ґрунтуючись на значеннях видовий матриці і матриці перспективної проекції. Фрагментний шейдер використовується для обчислення кольору наших ліній.

```
attribute vec3 aVertexPosition;
attribute vec4 aVertexColor;
uniform mat4 mvMatrix;
uniform mat4 prMatrix;
varying vec4 color;

void main(void)
{
        gl_Position = prMatrix * mvMatrix * vec4 ( aVertexPosition, 1.0 );
        color       = aVertexColor;
}
```

Рисунок 3.2. Вершинний рейдер.

```
 #ifdef GL_ES
   precision highp float;
#endif
varying vec4 color;
void main(void)
{
    gl_FragColor = color;
}
```

Рисунок 3.3. Фрагментний рейдер

Модельна видова матриця і матриця перспективної проекції

Обидві матриці мають розмір 4x4 і використовуються для розрахунку відображення тривимірних об'єктів на двовимірну площину. Їх відмінність в тому, що видова матриця визначає, як об'єкти виглядатимуть для спостерігача, наприклад, при зміні його положення, а матриця проекції спочатку задає спосіб проектування.

У програмі значення матриці проекції задаються при виклику функції gluPerspective на етапі ініціалізації, в подальшому ця матриця не змінює своїх значень. Функція gluPerspective не є стандартною. Її аргументи: fovy, aspect, zNear, zFar. fovy - область кута перегляду по вертикалі в градусах; aspect - відношення ширини області перегляду до висоти; zNear - відстань до ближньої площини відсікання; zFar - відстань до далекої площині відсікання.

Для задання значень модельно-видової матриці можна використовувати кілька підходів. Наприклад, створити і використовувати функцію gluLookAt (camX, camY, camZ, tarX, tarY, tarZ, upX, upY, upZ) - аналог функції для OpenGL, яка приймає в якості аргументів координати положення камери, координати цілі камери і up-вектор камери. Інший спосіб, це створення і використання функцій glTranslate, glRotate, glScale, які виробляють зрушення, обертання, масштабування відносно спостерігача (камери). Для первинного визначення положення камери можна використовувати gluLookAt, а для наступних перетворень використовувати glTranslate, glRotate, glScale. Так чи інакше, ці функції лише змінюють значення однієї і тієї ж модельно-видової матриці. Для зручності обчислення матриць можна використовувати бібліотеку sylvester.js.

У вершинному шейдері для модельно-видової матриці використовується змінна «mvMatrix». Щоб передати цій змінній значення матриці, потрібно спочатку отримати її індекс в програмі. Для цього використовується функція loc = gl.getUniformLocation (shaderProgram, name), яка є стандартною. Перший аргумент - змінна, яка вказує на програму шейдеров, яка отримана на другому етапі, а аргумент «name» - ім'я змінної, якій ми хочемо передати значення, в нашому випадку name = "mvMatrix». Тепер, отримавши індекс, використовуємо функцію gl.uniformMatrix4fv (loc, false, new Float32Array (mat.flatten ())) для передачі значення матриці mat. Аналогічно, отримується індекс і встановлюється значення для матриці проекції. Видову матрицю в шейдерній програмі потрібно оновлювати щоразу при зміні її значень, щоб вони вступили в силу.

Буфери даних

Використання буферів в WebGL обов'язково. Положення кожної точки і її колір потрібно зберігати в двох буферах.

```
vPosBuffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, vPosBuffer);

gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(verticies), gl.DYNAMIC_DRAW);


gl.vertexAttribPointer(vPosLoc, 3, gl.FLOAT, false, 0, 0);

gl.enableVertexAttribArray(vPosLoc);
```

Рисунок 3.4. Координати точок для буфера.

Очищення буфера кольору або, створення фону.

```
gl.clearColor(0, 0, 0, 1);
gl.clear(gl.COLOR_BUFFER_BIT);
```

Рисунок 3.5. Створення фону.

```
gl.drawArrays(gl.LINES, drawArrayIndexStart, drawArrayLength);
```

Рисунок 3.6. Малювання.

Перший аргумент цієї функції – відображення ліній, другий - індекс в буфері, з якого почнеться відображення, drawArrayLength - кількість елементів для малювання.

У порівнянні з іншими технологіями WebGL має ряд переваг:

- JavaScript програмування. Робота з JavaScript дозволяє отримати доступ до всіх DOM-елементів, а також легко з ними звертатися, в відміну від спілкування з апплетами. Так як WebGL програмується в JavaScript, то це полегшує інтеграцію WebGL-додатків з іншими

JavaScript - бібліотеками, такими як JQuery і іншими технологіями HTML5.

- Автоматичне управління пам'яттю: на відміну від свого побратима OpenGL і інших технологій, де є конкретні операції виділення і звільнення пам'яті вручну, в WebGL немає такої необхідності. З цього випливає, що при виході JavaScript змінної з області видимості, пам'ять, яку займає їй, автоматично звільняється. Це надзвичайно полегшує програмування, зменшує обсяг коду, робить його більш ясним і зрозумілим.
- Проникність: завдяки сучасним технологічним досягненням, веб-браузери з підтримкою JavaScript встановлюються на смартфони і планшетні пристрої. На момент написання, Mozilla Foundation є програмою для тестування можливостей WebGL в телефонах Motorola і Samsung. Існують подібні розробки підтримки WebGL для платформи Android.
- Продуктивність: продуктивність додатків WebGL порівнянна з еквівалентними автономними додатками (з деякими винятками). Це відбувається завдяки здатності WebGL мати доступ до локальних апаратним прискорювачів графіки. До сих пір, багато веб-технології для 3D візуалізації використовують програмний рендеринг.
- Нульова компіляція: враховуючи, що WebGL написана на JavaScript, то немає необхідності в попередньої компіляції коду перед виконанням в веб-браузері. Це дозволяє вносити зміни на льоту і дивитися, як ці зміни впливають на 3D веб-додаток. Проте, коли ми торкнемося теми шейдеров, то ми зрозуміємо, що потребуємо деякої компіляції. Однак, це відбувається за допомогою наших графічних апаратних засобів, а не в нашому браузері.

3.2. Three JS

Сама бібліотека написана на JavaScript і призначена для використання в середовищі javascript. Здебільшого це означає, що він буде працювати на стороні клієнта - у веб-браузері на якомусь пристрої. Але з node.js та безголовними браузерами це також може використовуватися на стороні сервера.

Більшість проектів three.js, містять 3D-графіку в реальному часі, де взаємодія користувача призводить до негайного візуального зворотного зв’язку. Інший тип 3D-графіки - це або різні ефекти, або штучні персонажі у фільмах, або різноманітні «візуалізації», які ви можете побачити надрукованими або у веб-оглядачі.

Підмножиною всього цього є 3d математика. 3D-графіку неможливо зробити без математики, а комп'ютерні мови не розуміють 3d-концепції за замовчуванням. Тут надходить бібліотека, вона резюмує ті математичні операції, можливо, оптимізує їх та відкриває інтерфейс високого рівня, такий як Matrix4 або .dot ().

Three.js має власну математичну бібліотеку з певними класами для 3d математики. Є автономні бібліотеки, які займаються цією математикою самостійно, але з трьома, це лише підмножина значно більшої системи.

Граф сцени

Можна виділити область з three.js, яка служить цією загальною абстракцією "3d світу". Граф сцени - це структура даних, яка використовується для опису того, як об'єкти в деякій 3d сцені (світі) співвідносяться один з одним. Насправді це не повинно бути 3d, оскільки це підходящий спосіб описати будь-яку векторну графічну ієрархію. Це конкретно "дерево", виготовлене з "вузлів" із "кореневим вузлом", який розгалужується. У three.js базовим класом для цієї структури даних є Object3D.

Це майже точно так само, як дерево DOM. Сцена була б аналогічною <body>, а все інше - це галузі. У DOM ми можемо позиціонувати речі, але досить

обмежені. Обертання зазвичай відбувається навколо однієї осі, і ми переміщуємо речі вліво / вправо або вгору / вниз.

Сцена Three нагадує віртуальний DOM. Ми робимо наші операції і встановлюємо стан на цьому дереві, і коли нам хочеться зробити візуальний знімок цього стану (скажімо, у безперервному циклі або деяка взаємодія користувача / зміна стану), ми викликаємо рендер (сцену). Ви не хочете оновлювати ціле дерево DOM, коли щось змінюється, тоді як за допомогою елемента <canvas> ми повинні очистити весь погляд, а потім перемалювати все, навіть якщо лише один елемент змінив позицію.

3.3. Babylon JS

Бібліотека з відкритим вихідним кодом для створення повноцінних 3D додатків та ігор, які працюють в веб-браузері без використання сторонніх плагінів і розширень. Babylon JS за своїми можливостями близька до ThreeJS, проте має в своєму арсеналі деякі вбудовані функції, недоступні в Three JS з коробки. До таких приємних особливостей відносяться вбудований фізичний движок oimo.js - досить простий спосіб створити реалістичний ландшафт, використовуючи карту висот. Зрозуміло, в three js також присутні такі можливості, але реалізовані вони за допомогою різних додаткових додатків. Однак за функціональність бібліотеки доводитися платити нескромним вагою в 800 кб.

Babylon.js створений з використанням мови TypeScript. TypeScript компільований і мультиплатформенний мову генерує чистий JavaScript код.

3.4. Порівнянна Three JS та Babylon JS

3.4.1. Базові елементи

Сцена:

У THREE.JS створення сцени відбувається з додавання в document.body renderer.domElement.

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

Рисунок 3.7. Додавання сцени на сторінку в Three JS.

У BABYLON.JS не може бути контейнером для сцени будь div наприклад. Все починається зі створення елемента canvas:

```
<canvas id="renderCanvas"></canvas>
```

Рисунок 3.8. Додавання сцени на сторінку в Babylon JS.

Як і у three.js також опцією включається antialiasing.

```
var canvas = document.getElementById("renderCanvas");
var engine = new BABYLON.Engine(canvas, true);
```

Рисунок 3.9. Включення antialiasing.

Створення самої сцени:

BABYLON.JS

scene отримує параметром engine.

```
scene = new BABYLON.Scene(engine);
```

Рисунок 3.10. Створення сцени в Babylon JS.

THREE.JS

Сцена створюється як би окремо. І в неї вже додаються всі елементи сцени.

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

Рисунок 3.11. Створення сцени в Three JS.

Після цього в THREE.JS renderer викликається в будь-який функції з наявністю requestAnimationFrame найчастіше називають animate або render, а в BABYLON.JS колбек в engine.runRenderLoop.

У THREE.JS, найчастіше в animate додається вся логіка рухів, наприклад, політ куль, обертання об’єктів, біганина ботів тощо.

```
function animate() {
    requestAnimationFrame(animate);
    renderer.render(scene, camera);
}
```

Рисунок 3.12. Створення animate функції.

Як це виглядає в BABYLON.JS, тут як правило іноді додають якісь загальні конструкції, підрахунок частоти кадрів, вершин, кількості частинок та інше. Простіше кажучи, статистику. Для різних анімацій є гарний хук, докладніше про це в розділі про анімацію.

```
engine.runRenderLoop(function () {
        scene.render();
        stats.innerHTML = "FPS: <b>" +  BABYLON.Tools.GetFps().toFixed() + "</b>
});
```

Рисунок 3.13. Створення колбек функції.

Примітиви:

Після ініціалізації сцени перше, що можна зробити - це створити примітиви.

Тут все схоже у babylon.js виглядає компактніше додавання на сцену об'єкта просто опцією, а у three.js прості маніпуляції з призначеннями матеріалів виглядають компактніше.

BABYLON.JS

```
var sphere = BABYLON.Mesh.CreateSphere("sphere1", 16, 2, scene);
sphere.material = new BABYLON.StandardMaterial("texture1", scene);
sphere.material.diffuseColor = new BABYLON.Color3(1, 0, 0); //красный
sphere.material.alpha = 0.3;
```

Рисунок 3.14. Створення примітивної фігури в Babylon JS.

THREE.JS

```
var cube = new THREE.Mesh( new THREE.BoxGeometry( 1, 1, 1 ), new THREE.MeshBasicMaterial({ colo
r: 0x00ff00 }) );
scene.add( cube );
```

Рисунок 3.15. Створення примітивної фігури в Three JS.

Позиція для координат окремо вказується однаково:

```
mesh.position.x = 1;
mesh.position.y = 1;
mesh.position.z = 1;
```

Рисунок 3.16. Позиції координат.

А щоб відразу поставити є відмінності, наприклад в THREE.JS можна написати ось так:

```
mesh.position.set(1, 1, 1);
mesh.rotation.set(1, 1, 1);
```

Рисунок 3.17. Позиції координат в Three JS.

А в BABYLON.JS в основному так:

```
mesh.position = new BABYLON.Vector3(1, 1, 1);
```

Рисунок 3.18. Позиції координат в Babylon JS.

Камери:

У обох бібліотек найбільш використовуваних камер по дві, хоча є додаткові у babylon.js наприклад є з різними фільтрами і спеціально для планшетів та інших девайсів. Спеціально для цього зазвичай ще потрібно підключати hand.js

BABYLON.JS

FreeCamera - Показує перспективну проекцію, але з можливістю призначати клавіші для управління, що зручно використовувати в іграх від першої особи, докладніше в розділі про Пересування персонажа

ArcRotateCamera - Камера передбачає обертання навколо заданої осі, за допомогою курсору миші або сенсора, якщо попередньо підключити hand.js

THREE.JS

PerspectiveCamera - Камера перспективної проекції, трохи спрощений аналог FreeCamera. Залежить від пропорцій і поля зору і показує реальний світ.

OrthographicCamera - Камера ортогональної проекції, показує всі об'єкти сцени однаковими, без пропорцій.

Обертати мишкою сцену в three.js допомагає плагін OrbitControls.js. У babylon.js схожа можливість є в ArcRotateCamera.

```
new THREE.PerspectiveCamera( 45, width / height, 1, 1000 );
new THREE.OrthographicCamera( width / - 2, width / 2, height / 2, height / - 2, 1, 1000 );
```

Рисунок 3.19. Налаштування OrthographicCamera в Three JS.

З додаткового є CombinedCamera - дозволяє встановлювати фокусну відстань об'єктива і перемикатися між перспективною і ортогональної проекціями.

```
new THREE.CombinedCamera( width, height, fov, near, far, orthoNear, orthoFar )
```

Рисунок 3.20. Налаштування CombinedCamera в Three JS.

Освітлення:

BABYLON.JS

- Point Light - Точкове світло, імітує світлове пляма.
- Directional Light - Спрямований трохи розсіяне світло.
- Spot Light - Більше схожий на імітацію ліхтарика, наприклад може імітувати рух світила.
- HemisphericLight - підходить для імітації реалістичною навколишнього середовища, рівномірно освітлює.
-

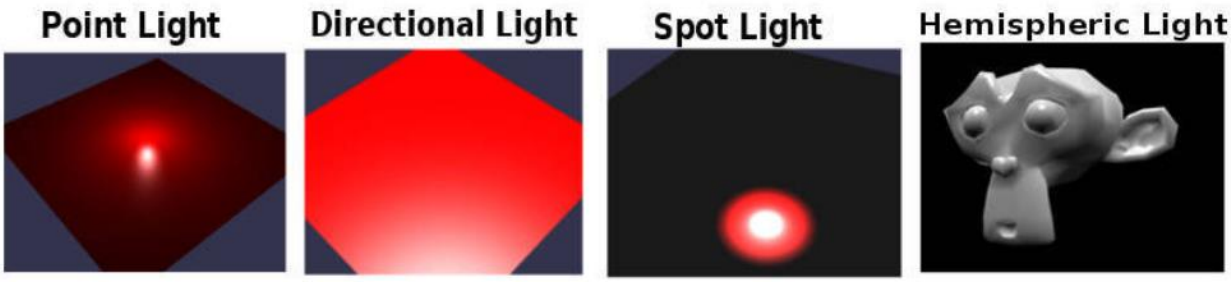

Рисунок 3.21. Приклади освітлень в Babylon JS.

```
//Point Light
new BABYLON.PointLight("Omni0", new BABYLON.Vector3(1, 10, 1), scene);
//Directional Light
new BABYLON.DirectionalLight("Dir0", new BABYLON.Vector3(0, -1, 0), scene);
//Spot Light
new BABYLON.SpotLight("Spot0", new BABYLON.Vector3(0, 30, -10), new BABYLON.Vector3(0, -1, 0),
0.8, 2, scene);
//Hemispheric Light
new BABYLON.HemisphericLight("Hemi0", new BABYLON.Vector3(0, 1, 0), scene);
```

Рисунок 3.22. Створення освітлень в Babylon JS.

THREE.JS

- AmbientLight - представляє загальне освітлення, яке застосовується до всіх об'єктів сцени.
- AreaLight представляє просторовий джерело світла, що має розміри - ширину і висоту і орієнтований в просторі.
- DirectionalLight - представляє джерело прямого (спрямованого) освітлення - потік паралельних променів в напрямку об'єкта.
- HemisphereLight - представляє напівсферичне освітлення.
- SpotLight - представляє прожектор.

```
//ambientLight
var ambientLight = new THREE.AmbientLight( 0x404040 );
//AreaLight
areaLight1 = new THREE.AreaLight( 0xffffff, 1 );
areaLight1.position.set( 0.0001, 10.0001, -18.5001 );
areaLight1.width = 10;
//DirectionalLight
var directionalLight = new THREE.DirectionalLight( 0xffffff, 0.5 );
directionalLight.position.set( 0, 1, 0 );
//PointLight
var pointLight = new THREE.PointLight( 0xff0000, 1, 100 );
pointLight.position.set( 50, 50, 50 );
//PointLight
var spotLight = new THREE.SpotLight( 0xffffff );
spotLight.position.set( 100, 1000, 100 );
```

Рисунок 3.23. Створення освітлень в Three JS.

Матеріали:

Підхід до матеріалів вже досить сильно різниться, якщо у three.js є список можливих матеріалів, то у babylon.js по суті є тільки один матеріал і до нього застосовуються різні властивості: прозорість, накладання текстур з подальшим їх зміщенням по осях і тому подібне.

BABYLON.JS

```
var materialSphere6 = new BABYLON.StandardMaterial("texture1", scene);
materialSphere6.diffuseTexture = new BABYLON.Texture("./tree.png", scene);
```

Рисунок 3.24. Створення матеріалу і призначення текстури.

```
var materialSphere2 = new BABYLON.StandardMaterial("texture2", scene);
materialSphere2.diffuseColor = new BABYLON.Color3(1, 0, 0); //Red
materialSphere2.alpha = 0.3;
```

Рисунок 3.25. Створення матеріалу і призначення йому кольору і прозорості.

Рисунок 3.26. Приклади матеріалів на об’єктах.

THREE.JS

- MeshBasicMaterial - просто призначає будь-який колір примітиву.
- MeshNormalMaterial - матеріал з властивостями shading, поєднує в собі змішання кольорів.
- MeshDepthMaterial - матеріал з властивостями wireframe, виглядає чорно-білим.

- MeshLambertMaterial - матеріал для не блискучих поверхонь.
- MeshPhongMaterial - матеріал для блискучих поверхонь.
- MeshFaceMaterial - може комбінувати інші види матеріалів призначати на кожен полігон свій матеріал.

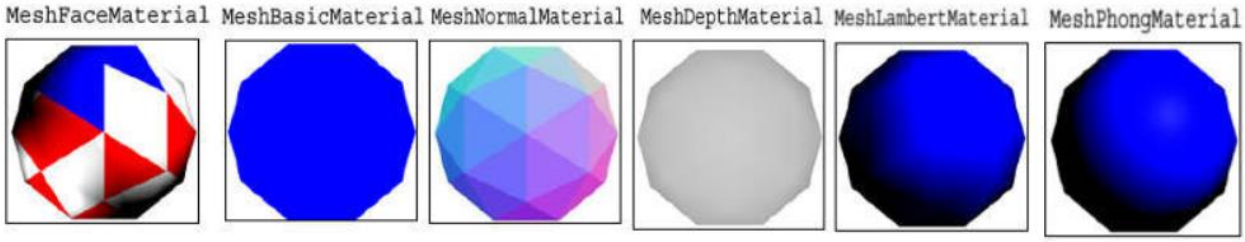

Рисунок 3.29. Приклади матеріалів в Three JS.

3.4.2. Угруповання, рух та частинки.

BABYLON.JS

Є кілька способів угруповання. Найпростіший і очевидний - це призначення властивості parent. Наприклад, якщо треба закріпити якийсь об'єкт за камерою:

```
var mesh = new BABYLON.Mesh.CreateBox('name', 1.0, scene);
mesh.position = new BABYLON.Vector3( 1, -1, 5);
mesh.parent = camera;
```

Рисунок 3.30. Закріплення об’єкта за камерою.

У THREE.JS нам треба створити батьківський об'єкт для всіх примітивів і в нього помістити інші об'єкти, а потім вже управляти цим батьківським об'єктом на свій розсуд:

```
var parent = new THREE.Object3D();
parent.add( camera );
parent.add( mesh );
parent.position.y = 10;
```

Рисунок 3.31. Управління камерою від першої особи.

Для ігор від першої особи потрібно щоб камера, що показує перспективу, управлялася мишкою і з клавіатури.

BABYLON.JS

FreeCamera дозволяє відразу керувати рухом. Досить вказати camera.detachControl (canvas). Камері можна задати ряд властивостей, наприклад, швидкість camera.speed = 1, призначити клавіші «вперед», «назад», «вліво», «вправо» і т.д.

```
camera.keysUp =       [38, 87];
camera.keysDown =  [40, 83];
camera.keysLeft =     [37, 65];
camera.keysRight =   [39, 68];
```

Рисунок 3.32. Управління камерою від першої особи за допомогою клавіатури.

Слід зауважити, що повноцінне керування мишею включається тільки після підключення PointerLock. Камері можна привласнити дітей, які будуть разом з нею їздити. Відповідно, якщо ми пишемо багато користувачів гру, то краще передавати координати камери camera.position, camera.cameraRotation сервера для управління.

А ось з THREE.JS все набагато складніше. Камера сама по собі тут просто камера і щоб змусити її рухатися потрібно прописувати на кожне натискання клавіші зміну позиції.

Для управління мишею теж все непросто - міняти, підставляти координати в mesh.rotation.set (x, y, z) явно недостатньо. Тут нас трохи рятує приклад з github.io автора three.js. Тому зупинюся лише на парі деталей. Щоб обертати мишею камеру з об'єктом потрібно спочатку створити один new THREE.Object3D (), всередину нього помістити інший і всередині обертати. Тоді вийде видимість повороту навколо своєї осі. Виглядає це все в скороченому варіанті приблизно так:

```
var pitchObject = new THREE.Object3D();
pitchObject.add( camera );
var yawObject = new THREE.Object3D();
yawObject.position.y = 10;
yawObject.add( pitchObject );
var onMouseMove = function ( event ) {
        yawObject.rotation.y -= event.movementX * 0.002;
        pitchObject.rotation.x -= event.movementY * 0.002;
        pitchObject.rotation.x = Math.max( - Math.PI / 2, Math.min( Math.PI / 2, pitchObject.ro
tation.x ) );
};
document.addEventListener( 'mousemove', onMouseMove, false );
```

Рисунок 3.33. Управління камерою від першої особи за допомогою миші.

У BABYLON.JS система частинок вбудована і рясніє великою кількістю налаштувань. Але для відтворення будь-якого ефекту необхідно експериментувати або бігати по форумам шукати сподобався. Плюс треба прив'язувати до готового Мешу. Його, звичайно, можна зробити невидимим, але можна було б просто зробити щоб була можливість просто вказати координати місцезнаходження.

У THREE.JS підключається за допомогою стороннього плагіна, але має можливість відразу відтворювати заготовлені ефекти і виставляти час. Місце появи частинок можна задати координатами.

3.4.3. Анімація, ландшафт та колізії.

Як правило, анімація потрібна для відтворення будь то ефектів, наприклад, руху небесних світил, руху ботів і тому подібного. Є кілька варіантів як змусити рухатися різні об'єкти, розглянемо їх по порядку.

BABYLON.JS

Вставити анімацію можна в будь-яке місце стандартним для бібліотеки способом:

```
scene.registerBeforeRender(function () {
        mesh.position.x = 100 * Math.cos(alpha);
        donutmesh.position.y = 5;
        mesh.position.z = 100 * Math.sin(alpha);
        alpha += 0.01;
});
```

Рисунок 3.34. Спосіб вставити анімацію в Babylon JS.

THREE.JS

Для three.js є можливість змусити щось рухатися тільки в петлі анімації:

```
var render = function () {
        requestAnimationFrame( render );
        cube.rotation.x += 0.1;
        cube.rotation.y += 0.1;
        renderer.render(scene, camera);
};
render();
```

Рисунок 3.35. Об’єкт рухається в петлі анімації.

Крім заздалегідь невизначеною анімації коли не знаєш що куди побіжить, є анімація певна, коли, наприклад, персонаж при певних обставинах робить заздалегідь відомі два-три кроки або зімітувати, наприклад, стрілянину автомата.

BABYLON.JS

```
var animationBox = new BABYLON.Animation("tutoAnimation", "scaling.x", 30, BABYLON.Animation.AN
IMATIONTYPE_FLOAT,  BABYLON.Animation.ANIMATIONLOOPMODE_CYCLE);

var keys = [];
keys.push({ frame: 0,  value: 1  });
keys.push({ frame: 20,  value: 0.2  });
keys.push({   frame: 100,   value: 1  });

animationBox.setKeys(keys);
box1.animations.push(animationBox);

scene.beginAnimation(box1, 0, 100, true);
```

Рисунок 3.36. Анімація зі зміною розміру боксу.

Часто потрібно створити якийсь невигадливий навколишній ландшафт, невеликі пагорби або гори. І щоб це було швидко.

BABYLON.JS

Відбувається шляхом поєднання однією оригінальною картинки ландшафту, яка буде фоном, і інший чорно-білої - виходить ніби видавлювання.

```
    var groundMaterial = new BABYLON.StandardMaterial("ground", scene);
    groundMaterial.diffuseTexture = new BABYLON.Texture("./img/earth.jpg", scene);
    var ground = BABYLON.Mesh.CreateGroundFromHeightMap("ground", "./img/heightMap.jpg", 200, 2
00, 250, 0, 10, scene, false);
    ground.material = groundMaterial;
```

Рисунок 3.37. Створення ландшафту в Babylon JS.

Рисунок 3.38. Приклад ландшафту в Babylon JS.

THREE.JS

```
var geometry    = THREEx.Terrain.heightMapToPlaneGeometry(heightMap)
THREEx.Terrain.heightMapToVertexColor(heightMap, geometry)
var material    = new THREE.MeshPhongMaterial({ shading :THREE.SmoothShading, vertexColors :THR
EE.VertexColors});
var mesh    = new THREE.Mesh( geometry, material );
scene.add( mesh );
```

Рисунок 3.39. Створення ландшафту в Three JS.

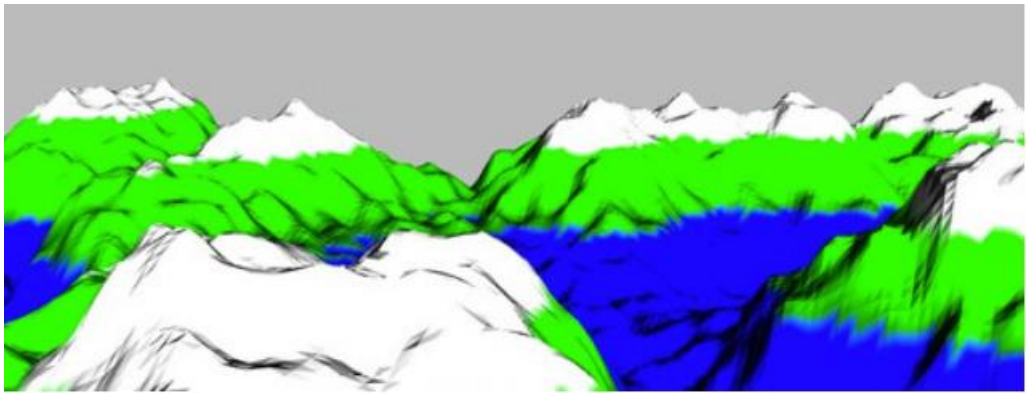

Рисунок 3.40. Приклад ландшафту в Babylon JS.

Зіткнення об'єктів заздалегідь передбачені.

BABYLON.JS

На кожен об'єкт можна виставити:

Але потрібно зауважити, що, наприклад, виставляти checkCollisions на велику кількість об'єктів або на об'ємну площа марно, все буде гальмувати.

А навколо ландшафтних вигинів краще вибудовувати коридори з невидимих примітивів.

Використовуються коли ви ніяк точно не можете знати, чи відбудеться зіткнення якогось одного об'єкта з іншим. А якщо станеться, то потрібно якось на це реагувати.

Все ідентично, практично. Приклади попадання кульок імітують потрапляння кулі в об'єкти.

BABYLON.JS

```
scene.registerBeforeRender(function () {
     for (var i=0; i < meshList.length; i++ ){
           if(bullet.intersectsMesh(meshList[i], true))
                console.log('Есть попадание, координаты:' meshList[i].position);
      }
});
```

Рисунок 3.41. Динамічна колізія в Babylon JS.

THREE.JS

```
function animate() {
       requestAnimationFrame(animate);
        for(var res = 0; res < meshList.length; res++) {
            var intersections = raycaster.intersectObject(meshList[res]);
            if (intersections.length > 0)
                console.log('Есть попадание, координаты:' ballMeshes[i].position);
    }
        renderer.render(scene, camera);
}
```

Рисунок 3.42. Динамічна колізія в Babylon JS.

Висновки до розділу

В цьому розділі розглянуто використання WebGL API, який є базою для графічних бібліотек. За допомогою WebGL також можливе створення графіки в браузерах, але це стає дедалі складніше ніж потрібно для очікуваного результату. Наведено системні вимоги, принцип та структура роботи WebGL додатку. Також наведено порівняння з іншими технологіями.

Також було проаналізовано два найпопулярніші та конкуруючі бібліотеки, які засновані на базі WebGL. Проведено порівняння у вигляді пошагового створення базових елементів.

4. РОЗРОБКА ТРИВИМІРНОЇ ГРИ З ВИКОРИСТАННЯМ ГРАФІЧНОЇ БІБЛІОТЕКИ THREE.JS

Створення простої 3D-гри літачок з допомогою бібліотеки Three.js, що спрощує роботу WebGL. WebGL - для багатьох темний ліс через складність і синтаксис GLSL. Але завдяки Three.js реалізація 3D в браузері є набагато простішою.

HTML і CSS

Спершу потрібно імпортувати бібліотеку в заголовок HTML:

```
<script src="./three-js.lib.js"></script>
```

Потім додати елемент-контейнер для зберігання сцени:

```
<div id="game-world-container"></div>
```

Можна стилізувати контейнер наступним чином:

```
#game-world-container {
    background:linear-gradient(rgb(228,224,186),
rgb(247,217,170));
    position: absolute;
    overflow: hidden;
    height: 100%;
    width: 100%;
}
```

На цьому етапі з використанням HTML та CSS покінчено.

JAVASCRIPT

Three.js дуже легко використовувати, якщо є базові знання по JavaScript.

По-перше потрібно визначитися з палітрою кольорів:

```
const COLORS_MAP = {
    red: “#F25346”,
    white: “#D8D0D1”,
    brown: “#59332E”,
    // інші кольори
};
```

Структура коду

Хоча JavaScript-код і буває громіздким, його структура досить проста. Всі основні дії потрібно помістити в функцію globalGameInit:

```
window.addEventListener('load', globalGameInit);
const globalGameInit = (event) => {
    // налаштування сцени, камери та відтворення
    createGlobalGameScene();
    // додавання світла
    createGlobalGameLights();
    // додавання об’єктів
    createGlobalGamePlane();
    createGlobalGameSea();
```

```
    createGlobalGameSky();
```

// запуск циклу що оновлює позиції об’єктів та відтворення сцени у кожному кадрі

```
    globalGameLoop();
}
```

4.1. Налаштування сцени

Для створення проекту на Three.js потрібен такий мінімальний набір:

Сцена: як майданчик, на який потрібно додати кожен об'єкт для відтворення.

Камера: використаємо перспективну камеру, можна взяти і ортогональну.

Рендерер: відображає всю сцену, використовуючи WebGL.

Один або декілька об'єктів: літак, море і небо (декілька хмар).

Один або кілька джерел світла: є декілька видів джерел. Використаємо півсферичне джерело для атмосфери і спрямоване - для тіней.

Налаштування сцени:

```
let globalGameScene, globalGameCamera, globalGameFieldOfView, globalGameAspectRatio, globalGameNearPlane, globalGameFarPlane, globalGameRenderer, globalGameContainer;


const createGlobalGameScene = () => {
```

// Потрібно зчитати висоту та ширину екрану, щоб застосувати їх для обчислення відношення сторін та розміру відтворювача.

```
GAME_HEIGHT = globalWindow.innerHeight;

GAME_WIDTH = globalWindow.innerWidth;
```

// Створення сцени

```
globalGameScene = new ThreeJSGL.Scene();
```

// Створення камери

```
globalGameAspectRatio = GAME_WIDTH / GAME_HEIGHT;

globalGameFieldOfView = 60;

globalGameNearPlane = 1;

globalGameFarPlane = 10000;

globalGameCamera = new ThreeJSGL.PerspectiveCamera(

globalGameFieldOfView,          globalGameAspectRatio,
globalGameNearPlane, globalGameFarPlane);
```

// Додавання ефекту «fog»;

```
globalGameScene.fog  =  new  ThreeJSGL.Fog(#F7D9AA,  90,
850);
```

// Встановлення позиції камери

```
globalGameCamera.position.x = 0;

globalGameCamera.position.z = 200;

globalGameCamera.position.y = 100;
```

// Створення механізму відтворення

```
    globalGameRenderer = new ThreeJSGL.WebGLRenderer({
alpha: true, antialias: true });

    // Встановлення розміру відтворювача. Відтворювач повинен бути такого ж розміру як і екран

    globalGameRenderer.setSize(GAME_WIDTH, GAME_HEIGHT);

    // Ввімкнення відтворення тіней

    globalGameRenderer.shadowMap.enabled = true;

    // Додавання до DOM вузла гри

    globalGameContainer = globalDocument.getElementById('game-world-container');

    globalGameContainer.appendChild(globalGameRenderer.domElement);

    // Прослуховування події зміни розміру вікна

    globalWindow.addEventListener('resize', handleGlobalWindowResize);

}
```

Оскільки розмір екрану може змінюватися, потрібно оновлювати розмір відтворювача і співвідношення сторін камери:

```
const handleGlobalWindowResize = () => {

    // оновлення розміру відтворювача та співвідношення сторін камери

    GAME_HEIGHT = globalWindow.innerHeight;

    GAME_WIDTH = globalWindow.innerWidth;

    globalGameRenderer.setSize(GAME_WIDTH, GAME_HEIGHT);
```

```
    globalGameCamera.aspect = GAME_WIDTH / GAME_HEIGHT;
    globalGameCamera.updateProjectionMatrix();
}
```

Джерела світла

Налаштування освітлення - це один з найскладніших етапів створення сцени. Джерела світла задають настрій всієї сцени, тому працювати з ними потрібно дуже уважно. На даному етапі потрібно просто налагодимо освітлення таки чином, щоб бачити всі об'єкти.

```
    let ambientGlobalLight, hemisphereGlobalLight, shadowGlobalLight;
const createGlobalGameLights = () => {
    // Світло півкулі - це світло градієнтного кольору
    hemisphereGlobalLight = new ThreeJSGL.HemisphereLight(0xaaaaaa,0x000000, .9)
    // Спрямоване світло світить з певного напрямку
    shadowGlobalLight = new ThreeJSGL.DirectionalLight(0xffffff, .9);
    shadowGlobalLight.position.set(150, 350, 350);
    // Лиття тіней
    shadowGlobalLight.castShadow = true;
    // Створення видимої області проектованої тіні
    shadowGlobalLight.shadow.camera.left = -400;
    shadowGlobalLight.shadow.camera.right = 400;
```

```
    shadowGlobalLight.shadow.camera.top = 400;
    shadowGlobalLight.shadow.camera.bottom = -400;
    shadowGlobalLight.shadow.camera.near = 1;
    shadowGlobalLight.shadow.camera.far = 1000;
    // Визначення роздільної здатності тіні
    shadowGlobalLight.shadow.mapSize.width = 2048;
    shadowGlobalLight.shadow.mapSize.height = 2048;
    // Додавання ліхтарів до сцени
    globalGameScene.add(hemisphereGlobalLight);
    globalGameScene.add(shadowGlobalLight);
}
```

Видно, що для налаштування освітлення використовується багато параметрів. Необхідно постійно експериментувати з кольорами, їх інтенсивністю і джерелами світла. Таким чином можна виявити величезну кількість цікавих поєднань і розібратися, як їх налаштовувати відповідно до потреб.

4.2. Створення об’єкту.

В даній частині, можливе використання імпортованих 3D-моделей. Але потрібно розглянути той випадок коли немає готової моделі. Створення моделі-об'єкту з примітивів Three.js, для кращого розуміння бібліотеки

У Three.js доступні наступні примітиви: куб, сфера, тор, циліндр і площину. В проекті всі об'єкти будуть комбінаціями цих тіл.

Простий циліндр для моря. Створення моря. Налаштування моря та хвиль.

```
// Визначення об’єкту моря:
const SeaConstructor = function () {
// Створення геометрії (форми) циліндра
    const geometry = new ThreeJSGL.CylinderGeometry(600, 600, 800, 40, 10);
// Обертання геометрії по осі x
    geometry.applyMatrix(new ThreeJSGL.Matrix4().makeRotationX(-MathPI / 2));
// Створення матеріалу
    const material = new ThreeJSGL.MeshPhongMaterial({
// параметри матеріалу
    });
// Щоб створити об’єкт у Three.js, необхідно створити сітку, яка є поєднанням геометрії та деякого матеріалу
    this.mesh = new ThreeJSGL.Mesh(geometry, material);
// Ввімкнення тіней
    this.mesh.receiveShadow = true;
}
    let seaInstance;
const createGlobalGameSea = () => {
// Створення екземпляру класу SeaConstructor
```

```
    seaInstance = new SeaConstructor();
    seaInstance.mesh.position.y = -600;

    // додавання сітки моря до сцени
    globalGameScene.add(seaInstance.mesh);
}
```

Отже, у підсумку, що потрібно для створення об'єкта:

- створити геометричну модель;
- створити матеріал;
- передати їх в меш;
- додати меш на сцену.

Дотримуючись цих кроків, можливо створити безліч різних примітивних об'єктів і об'єднати їх в більш складні фігури.

Об'єднання простих кубів для створення складної фігури.

Хмари вже трохи складніше моря, оскільки вони складаються з декількох з'єднаних випадковим чином кубів.

```
const CloudConstructor = function () {
    // створення контейнеру що буде зберігати різні частини хмари
    this.mesh = new ThreeJSGL.Object3D();
    this.mesh.name = "cloud";
    // створення геометричного кубу
    const geometry = new ThreeJSGL.CubeGeometry(20, 20, 20);
```

```
// Створення матеріалу
const material = new ThreeJSGL.MeshPhongMaterial({
// налаштування матеріалу
});
// створення копій геометрії
const numberOfBlocks = 3 + Math.floor(Math.random() * 3);
for (let i = 0; i < numberOfBlocks; i++){
// Створення нового контейнеру для копій геометрії
let blockClone = new ThreeJSGL.Mesh(geometry.clone(), material);
blockClone.position.y = Math.random() * 10;
blockClone.rotation.y = Math.random() * MathPI * 2;
blockClone.position.z = Math.random() * 10;
blockClone.rotation.z = Math.random() * MathPI * 2;
blockClone.position.x = i * 15;
// розмір кубу
let size = .1 + Math.random() * .9;
// Дозвіл кожному кубу створювати та отримувати тіні
blockClone.receiveShadow = true;
blockClone.castShadow = true;
blockClone.scale.set(size, size, size);
```

```
// додавання контейнеру копій до першого контейнеру
this.mesh.add(blockClone);
}
};
```

Тепер, після створення хмари, потрібно заповнити нею все небо, розмістивши її копії випадковим чином на осі z:

```
// Створення конструктор-класу хмари
const SkyConstructor = function () {
// створення пустого контейнеру
this.mesh = new ThreeJSGL.Object3D();
this.clouds = [];
// визначення кількості хмар
this.numberOfClouds = 20;
// Щоб розподіляти хмари послідовно, потрібно розмістити їх за рівномірним кутом
const stepAngle = MathPI * 2 / this.numberOfClouds;
// створення хмар
for (let i = 0; i < this.numberOfClouds; i++) {
// встановлення обертання та положення кожної хмари
let cloud = new Cloud(); // це заключний кут хмари
let h = 750 + Math.random() * 200; // відстань між центром осі та самою хмарою
```

```
// Перетворення полярних координат (кут, відстань) в декартові координати (x, y)
    let a = stepAngle * i;
    this.clouds.push(cloud);
    const sinA = Math.sin(a);
    const cosA = Math.cos(a);
    cloud.mesh.position.y = sinA * h;
    cloud.mesh.position.x = cosA * h;
// для кращого результату необхідно розмістити хмари у випадкових глибинах всередині сцени
    cloud.mesh.position.z = Math.random() * 400 - 400;
// встановлення обертання хмари відповідно до її положення
    cloud.mesh.rotation.z = a + MathPI / 2;
// Також потрібно встановити випадкову шкалу для кожної хмари
    let size = Math.random() * 2 + 1;
    cloud.mesh.scale.set(size, size, size);
// додавання сітки кожної хмари в сцені
    this.mesh.add(c.mesh);
    }
};
// створення екземпляр-класу SkyInstance
let skyInstance;
const createGlobalGameSky = () => {
```

```
    skyInstance = new SkyConstructor();
    skyInstance.mesh.position.y = -600;
    globalGameScene.add(skyInstance.mesh);
};
```

Створення літака

Весь процес полягає в комбінуванні і інкапсуляції фігур.

```
const AirPlaneConstructor = function () {
    this.airPlaneMesh = new ThreeJSGL.Object3D();
    this.mesh.name = "AirPlaneConstructor";
    // Створення кабіни
    const geometryCockpit = new ThreeJSGL.BoxGeometry(60, 50, 50, 1, 1, 1);
    const materialCockpit = new ThreeJSGL.MeshPhongMaterial({color: COLORS_MAP.red, shading: ThreeJSGL.FlatShading});
    const cockpit = new ThreeJSGL.Mesh(geometryCockpit, materialCockpit);
    cockpit.receiveShadow = true;
    cockpit.castShadow = true;
    this.airPlaneMesh.add(cockpit);
    // створення двигуна
    const geometryEngine = new ThreeJSGL.BoxGeometry(20, 50, 50, 1, 1, 1);
```

```
    const materialEngine = new ThreeJSGL.MeshPhongMaterial({color: COLORS_MAP.white, shading: ThreeJSGL.FlatShading});

    const engine = new ThreeJSGL.Mesh(geometryEngine, materialEngine);

    engine.position.x = 40;

    engine.receiveShadow = true;

    engine.castShadow = true;

    this.airPlaneMesh.add(engine);

    // створення хвоста

    const geometryTailPlane = new ThreeJSGL.BoxGeometry(15, 20, 5, 1, 1, 1);

    const materialTailPlane = new ThreeJSGL.MeshPhongMaterial({color: COLORS_MAP.red, shading: ThreeJSGL.FlatShading});

    const tailPlane = new ThreeJSGL.Mesh(geometryTailPlane, materialTailPlane);

    tailPlane.position.set(-35, 25, 0);

    tailPlane.receiveShadow = true;

    tailPlane.castShadow = true;

    this.airPlaneMesh.add(tailPlane);

    // створення крил

    const geometrySideWing = new ThreeJSGL.BoxGeometry(40, 8, 150, 1, 1, 1);
```

```
    const materialSideWing = new ThreeJSGL.MeshPhongMaterial({color: COLORS_MAP.red, shading: ThreeJSGL.FlatShading});

    const sideWing = new ThreeJSGL.Mesh(geometrySideWing, materialSideWing);

    sideWing.position.set(0, 0, 0);

    sideWing.receiveShadow = true;

    sideWing.castShadow = true;

    this.airPlaneMesh.add(sideWing);

    // створення пропеллера

    const geometryPropeller = new ThreeJSGL.BoxGeometry(20, 10, 10, 1, 1, 1);

    const materialPropeller = new ThreeJSGL.MeshPhongMaterial({color: COLORS_MAP.brown, shading: ThreeJSGL.FlatShading});

    this.planePropeller = new ThreeJSGL.Mesh(geometryPropeller, materialPropeller);

    this.planePropeller.receiveShadow = true;

    this.planePropeller.castShadow = true;

    const geometryBlade = new ThreeJSGL.BoxGeometry(1, 100, 20, 1, 1, 1);

    const materialBlade = new ThreeJSGL.MeshPhongMaterial({color: COLORS_MAP.brownDark, shading: ThreeJSGL.FlatShading});
```

```
    const bladeMesh = new ThreeJSGL.Mesh(geometryBlade,
materialBlade);
    bladeMesh.position.set(8, 0, 0);
    bladeMesh.receiveShadow = true;
    bladeMesh.castShadow = true;
    this.planePropeller.add(bladeMesh);
    this.planePropeller.position.set(50, 0, 0);
    this.airPlaneMesh.add(this.planePropeller);
    this.mesh = this.airPlaneMesh;
};
```

Розміщення літака на сцені:

```
    let airplaneInstance;
const createGlobalGamePlane = () => {
    airplaneInstance = new AirPlaneConstructor();
    airplaneInstance.mesh.scale.set(.25,.25,.25);
    airplaneInstance.mesh.position.y = 100;
    globalGameScene.add(airplaneInstance.mesh);
};
```

Відображення

```
    globalGameRenderer.render(globalGameScene,
globalGameCamera);
```

4.3. Анімація.

Налаштування обертання пропелера літака, моря і хмар.

Для цього знадобиться нескінченний цикл:

```
const globalGameLoop = () => {
    // Обертання гвинта, моря і неба
    airplaneInstance.propeller.rotation.x += 0.3;
    seaInstance.mesh.rotation.z += .005;
    skyInstance.mesh.rotation.z += .01;
    // відтворення сцени
    globalGameRenderer.render(globalGameScene,
globalGameCamera);
    // Виклик функції циклу знову
    requestAnimationFrame(globalGameLoop);
};
```

Виклик функції відображення винесено в цикл. Тому що кожну зміну об’єкта потрібно малювати заново

Налаштування взаємодії з мишею

Зараз літак розташований в центрі сцени. Але ж необхідно, щоб він слідував за мишею.

Після завантаження документа потрібно додати слухач рухів миші. Для цього потрібно змінити функцію globalGameInit наступним чином:

```
const globalGameInit = () => {
```

```
    // додавання слухача подій миші
    globalDocument.addEventListener('mousemove',
handlePlayerMouseMove);
    createGlobalGameScene();
    createGlobalGameLights();
    createGlobalGamePlane();
    createGlobalGameSea();
    createGlobalGameSky();
    globalGameLoop();
};
```

Крім того, слід створити нову функцію для обробки рухів миші:

```
let playerMousePosition = {
    x: 0,
    y: 0
};
    // обробка подій переміщення миші
const handlePlayerMouseMove = (event) => {
    // перетворення отриманого значення положення миші до нормованого значення, що змінюється від -1 до 1; це формула горизонтальної осі:
    const tx = -1 + (event.clientX / GAME_WIDTH)*2;
    // для вертикальної осі  потрібно обернути формулу тому що 2D-вісь у йде в протилежному напрямку від осі 3D у
```

```
    const ty = 1 - (event.clientY / GAME_HEIGHT)*2;
    playerMousePosition = {x:tx, y:ty};
};
```

Нормалізувавши положення миші можна коректно переміщати літак.

Знову змінимо цикл, додавши в нього функцію для оновлення положення літака:

```
const globalGameLoop = () => {
    // оновлення літака у кожному кадрі
    updatePlane();
    seaInstance.mesh.rotation.z += .005;
    skyInstance.mesh.rotation.z += .01;
    globalGameRenderer.render(globalGameScene,
globalGameCamera);
    requestAnimationFrame(globalGameLoop);
};


const updatePlane = () => {
```

// переміщення літака між -100 і 100 по горизонтальній осі, і від 25 до 175 по вертикальній осі, залежно від положення миші, яке коливається від -1 до 1 на обох осях; для досягнення цього слід використати функцію нормалізації

```
    const targetY = normalize(playerMousePosition.y,-
.75,.75,25, 175);
```

```
    const targetX = normalize(playerMousePosition.x,-
.75,.75,-100, 100);
    // оновлення положення літака
    airplaneInstance.mesh.position.y = targetY;
    airplaneInstance.mesh.position.x = targetX;
    airplaneInstance.propeller.rotation.x += 0.3;
};
const normalize = (v, vmin, vmax, tmin, tmax) => {
    const vTempMin = Math.min(v, vmax);
    const nv = Math.max(vTempMin, vmin);
    const dv = vmax - vmin;
    const pc = (nv - vmin) / dv;
    const dt = tmax - tmin;
    const tv = tmin + (pc * dt);
    return tv;
};
```

Тепер можна керувати літаком мишею.

4.4. Покращення візуальної якості сцени.

У цій частині необхідно звернути увагу на покращення візуальної якості сцени. Слід зробити рух літака набагато більш плавним і додати ефект хвиль.

```
// Створення кабіни
const geometryCockpit = new ThreeJSGL.BoxGeometry(80,50,50,1,1,1);
const materialCockpit = new ThreeJSGL.MeshPhongMaterial({color:COLORS_MAP.red, shading:ThreeJSGL.FlatShading});
// Отримання доступу до певної вершини форми через масив вершин, а потім переміщення його властивостей x, y і z:
geometryCockpit.vertices[4].y-=10;
geometryCockpit.vertices[4].z+=20;
geometryCockpit.vertices[5].y-=10;
geometryCockpit.vertices[5].z-=20;
geometryCockpit.vertices[6].y+=30;
geometryCockpit.vertices[6].z+=20;
geometryCockpit.vertices[7].y+=30;
geometryCockpit.vertices[7].z-=20;
const cockpitMesh = new ThreeJSGL.Mesh(geometryCockpit, materialCockpit);
cockpitMesh.receiveShadow = true;
```

```
cockpitMesh.castShadow = true;

this.mesh.add(cockpitMesh);
```

Приклад того, як змінювати фігури.

Створення пілота з зачіскою, яка розвивається на вітрі. Не складна задача коли йде мова про низькополігональні моделі.

```
const PilotConstructor = function () {

this.pilotMesh = new ThreeJSGL.Object3D();

this.pilotMesh.name = "pilotInstance";
```

// angleHairs - властивість, яка використовується для того, щоб анімувати волосся пізніше

```
this.angleHairs = 0;
```

// тіло пілота

```
const bodyGeometry = new ThreeJSGL.BoxGeometry(15, 15, 15);

const bodyMaterial = new ThreeJSGL.MeshPhongMaterial({color:COLORS_MAP.brown, shading:ThreeJSGL.FlatShading});

const bodyMesh = new ThreeJSGL.Mesh(bodyGeometry, bodyMaterial);

bodyMesh.position.set(2, -12, 0);

this.pilotMesh.add(bodyMesh);
```

// лице пілота

```
    const faceGeom = new ThreeJSGL.BoxGeometry(10, 10, 10);
    const faceMat = new ThreeJSGL.MeshLambertMaterial({color:COLORS_MAP.pink});
    const face = new ThreeJSGL.Mesh(faceGeom, faceMat);
    this.pilotMesh.add(face);
```

// волосся пілота

```
    const hairGeom = new ThreeJSGL.BoxGeometry(4, 4, 4);
    const hairMat = new ThreeJSGL.MeshLambertMaterial({color:COLORS_MAP.brown});
    const hair = new ThreeJSGL.Mesh(hairGeom, hairMat);
    hair.geometry.applyMatrix(new ThreeJSGL.Matrix4().makeTranslation(0,2,0));
```

// створення контейнеру для волосся

```
    const hairsObject = new ThreeJSGL.Object3D();
```

// Створення контейнера для волосків вгорі голови (ті, що будуть анімовані)

```
    this.hairsTop = new ThreeJSGL.Object3D();
```

// створення волосся у верхній частині голови та розміщення його на сітці 3 x 4

```
    for (let i = 0; i < 12; i++) {
    let h = hair.clone();
    let col = i%3;
    let row = Math.floor(i / 3);
```

```
    let startPositionZ = -4;

    let startPositionX = -4;

    h.position.set(startPositionX + row * 4, 0,
startPositionZ + col * 4);

    this.hairsTop.add(h);

    }

    hairsObject.add(this.hairsTop);
```

// Створення волосків збоку від обличчя

```
    const hairSideGeom = new ThreeJSGL.BoxGeometry(12, 4,
2);

    hairSideGeom.applyMatrix(new
ThreeJSGL.Matrix4().makeTranslation(-6, 0, 0));

    const hairSideRight = new ThreeJSGL.Mesh(hairSideGeom,
hairMat);

    const hairSideLeft = hairSideRight.clone();

    hairSideRight.position.set(8, -2, 6);

    hairSideLeft.position.set(8, -2, -6);

    hairsObject.add(hairSideRight);

    hairsObject.add(hairSideLeft);
```

// Створення волосків на потилиці

```
    const hairBackGeom = new ThreeJSGL.BoxGeometry(2, 8,
10);
```

```
    const hairBackMesh = new ThreeJSGL.Mesh(hairBackGeom, hairMat);

    hairBackMesh.position.set(-1, -4, 0);

    hairsObject.add(hairBackMesh);

    hairsObject.position.set(-5, 5, 0);

    this.pilotMesh.add(hairsObject);

    const glassGeometry = new ThreeJSGL.BoxGeometry(5, 5, 5);

    const glassMaterial = new ThreeJSGL.MeshLambertMaterial({color:COLORS_MAP.brown});

    const glassRight = new ThreeJSGL.Mesh(glassGeometry, glassMaterial);

    glassRight.position.set(6, 0, 3);

    const glassLeft = glassRight.clone();

    glassLeft.position.z = -glassRight.position.z;

    const glassAGeometry = new ThreeJSGL.BoxGeometry(11, 1, 11);

    const glassA = new ThreeJSGL.Mesh(glassAGeometry, glassMaterial);

    this.pilotMesh.add(glassRight);

    this.pilotMesh.add(glassLeft);

    this.pilotMesh.add(glassA);

    const earGeometry = new ThreeJSGL.BoxGeometry(2, 3, 2);
```

```
    const earLeft = new ThreeJSGL.Mesh(earGeometry, faceMat);
    earLeft.position.set(0, 0, -6);
    const earRight = earLeft.clone();
    earRight.position.set(0, 0, 6);
    this.pilotMesh.add(earLeft);
    this.pilotMesh.add(earRight);
    this.mesh  = this.pilotMesh;
    };
```

// рух волосся

```
    PilotConstructor.prototype.updateHairs = function(){
```

// об’єкт волосся

```
    const hairs = this.hairsTop.children;
```

// Оновлення їх відповідно до кута angleHairs

```
    const l = hairs.length;
    for (let i = 0; i < l; i++) {
    let h = hairs[i];
```

// кожен елемент волосся буде циклічно масштабуватися між 75% і 100% від свого початкового розміру

```
    const angleHairsCos = Math.cos(this.angleHairs + i / 3);
    h.scale.y = .75 + angleHairsCos * .25;
```

```
    }

    // збільшення кута для наступного кадру

    this.angleHairs += 0.16;

};
```

Щоб волосся рухалося, варто додати цей рядок в цикл:

```
airplaneInstance.pilot.updateHairs();
```

Для моря потрібні хвилі. Це можна реалізувати, використовуючи техніки, які були представлені раніше:

- переміщення вершин;
- циклічний рух кожної вершини.

Для створення хвиль слід обертати кожну вершину циліндра навколо її початкової позиції, задаючи їй випадкові кутову швидкість і радіус обертання.

Змінення моря:

```
const SeaConstructor = function(){

    const geometry = new ThreeJSGL.CylinderGeometry(600, 600, 800, 40, 10);

    geometry.applyMatrix(new ThreeJSGL.Matrix4().makeRotationX(-Math.PI / 2));

    // об’єднуючи вершини, забезпечується безперервність хвиль

    geometry.mergeVertices();

    // вершини

    const l = geometry.vertices.length;
```

```
    // створення масиву для зберігання нових даних, пов’язаних з кожною вершиною
    this.waves = [];
    for (let i = 0; i < l; i++) {
    // отримуємо кожну вершину
    let v = geometry.vertices[i];
    // збереження деяких пов’язаних з ним даних
    const vCoordinates = {y: vertices.y, x: vertices.x, z: vertices.z};
    this.waves.push({
    ...vCoordinates,
    ang: Math.random() * Math.PI * 2,
    amp: 5 + Math.random() * 15,
    speed: 0.016 + Math.random() * 0.032
    });
}
const material = new ThreeJSGL.MeshPhongMaterial({
    color: COLORS_MAP.blue,
    transparent: true,
    opacity: .8,
    shading: ThreeJSGL.FlatShading,
});
```

```
this.mesh = new ThreeJSGL.Mesh(geometry, material);
this.mesh.receiveShadow = true;
};
```

// тепер треба створити функцію, яка буде викликатися у кожному кадрі та оновлювати положення вершин для імітації хвиль

```
SeaConstructor.prototype.moveWaves = function () {
```

// вершини

```
const verts = this.mesh.geometry.vertices;
const l = verts.length;
for (let i = 0; i < l; i++) {
```

// отримання пов’язаних з ним даних

```
let v = verts[i];
let vprops = this.waves[i];
```

// оновлення положення вершини

```
let vpropsAngCos = Math.cos(vprops.ang);
let vpropsAngSin = Math.sin(vprops.ang);
v.x = vprops.x + vpropsAngCos * vprops.amp;
v.y = vprops.y + vpropsAngSin * vprops.amp;
```

// збільшення кута для наступного кадру

```
vprops.ang += vprops.speed;
}
```

```
    this.mesh.geometry.verticesNeedUpdate = true;

    seaInstance.mesh.rotation.z += .005;

};

    seaInstance.moveWaves();
```

Покращення освітлення сцени

У функції createGlobalGameLights слід додати:

// навколишнє світло, яке змінює глобальний колір сцени і робить тіні м'якшими

```
    ambientGlobalLight = new ThreeJSGL.AmbientLight(0xdc8874, .5);

    globalGameScene.add(ambientGlobalLight);
```

М'який політ

Цього можна досягти, додаючи в кожному кадрі частину відстані між літаком і метою. Загальний вигляд коду такий:

```
    currentPosition = currentPosition + (finalPosition - currentPosition) * fraction;
```

Для більшої реалістичності обертання літака також має змінюватися в залежності від напрямку руху. Якщо літак швидко піднімається, то він повинен швидко обертатися проти годинникової стрілки. Якщо він повільно опускається, то повинен повільно обертатися за годинниковою стрілкою.

Реалізація в функції updateGamePlane:

```
const updateGamePlane = () => {

    const targetY = normalize(playerMousePosition.y, -.75, .75, 25,  175);
```

```
    const targetX = normalize(playerMousePosition.x, -.75, .75, -100,  100);

    // Переміщення площини на кожному кадрі, додавши частку відстані, що залишилася

    airplaneInstance.mesh.position.y += (targetY-airplaneInstance.mesh.position.y) * 0.1;

    airplaneInstance.mesh.rotation.z = (targetY-airplaneInstance.mesh.position.y) * 0.0128;

    airplaneInstance.mesh.rotation.x = (airplaneInstance.mesh.position.y-targetY) * 0.0064;

    airplaneInstance.propeller.rotation.x += 0.3;

};
```

Тепер рух літака виглядає набагато елегантніше і реалістичніше. Змінюючи дробові величини, можна регулювати різкість рухів.

Висновки до розділу

У цьому розділі запропоновано підхід до створення графічних інтерфейсів. Визначено найкращий варіант реалізації в плані швидкості роботи, швидкості створення та загального результату для подальшого застосування.

Використання Three js бібліотеки у розробці графічних інтерфейсів має переваги у швидкості та якості розробки на відміну від всіх інших розглянутих технологій. Запронований підхіх не вимагає поглиблених знань OpenGL або WebGl, має можливості інтеграції з сучасними програмами для створення графічних моделей та може бути маштабованим в майбутньому.

5. СТАРТАП-ПРОЕКТ

5.1. Опис ідеї проекту

Таблиця 5.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Створення графічних інтерфейсів.	1. Освіта	Можливість студентам дізнатись більше про графічні інтерфейси.
	2. Кіно та відеоігри	Можливість реалізації створення відео та ігор з тривимірним персонажем
	3. Реклама	Залучення та зацікавленість абітурієнтів.

Опис до таблиці 5.2:

W – слабка сторона;

N – нейтральна сторона;

S – сильна сторона.

Таблиця 5.2 – Визначення характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W	N	S
		Запропонований метод	Bllend4Web	Verge3D			
1.	Об’єкти застосування	Наявність інтеграції з усіма графічними моделями	Інтеграція з моделями Blender	Інтеграція з моделями 3Ds Max			+
2.	Вартість додатку	безкоштовно	безкоштовно	300$		+	+

5.2. Технологічний аудит ідеї проекту.

Таблиця 5.3. Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Персональний онлайн сервіс	Спеціалізоване обладнання для Створення графічних інтерфейсів	Наявна	Доступна
		Застосування апаратних систем	Необхідно розробити	Доступна
		Розробка власних апаратно-програмних рішень	Наявна	Доступна

5.3. Аналіз ринкових можливостей запуску стартап-проекту

Таблиця 5.4. Попередня характеристика потенційного ринку стартап-проекту.

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	6
2	Динаміка ринку (якісна оцінка)	Зростає
3	Наявність обмежень для входу (вказати характер обмежень)	Зацікавлення потенційних клієнтів
4	Специфічні вимоги до стандартизації та сертифікації	Немає
5	Середня норма рентабельності в галузі (або по ринку), %	81%

Таблиця 5.5. Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Якісне програмне забезпечення, для роботи в інтернет мережі.	Люди, які прагнуть навчитися, студенти, ВНЗ, викладачі, великі корпорації.	Залежно від цільової групи програмне забезпечення може бути безкоштовним, або умовно безкоштовним.	- Надійність - Доступність - Простота - Зручність - Швидкість

Таблиця 5.6. Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Незацікавленість клієнтів	Внаслідок невдалого маркетингу клієнт може не зацікавитись послугами	Внесення додаткових сервісних послуг, та зниження цін
2	Втрата конкуренції	Втрата рангу надійного поставника	Якісне та кількісне нарощування інтенсивності та грамотна цінова політика

Таблиця 5.7. Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Перехід до домінування на ринку медійних послуг	Зростання попиту	Якісне та кількісне нарощування потужностей
2	Імплементація технологій в існуючі системи	Зростання попиту внаслідок зростання клієнтів	Якісне та кількісне нарощування потужностей

Таблиця 5.8. Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1.Чиста конкуренція	Використання схожих технологій	Стандартизація на високому рівні
2.Локальний	Відсутність єдиного національного постачальника послуг	Окремий підхід до кожної локальної ділянки
3.Міжгалузева	Відсутня	Відсутня
4.Товарно-видова	Застосування стандартизованих технологій	За необхідності, використання загальновживаних апаратних та програмних засобів
5.Цінова	Застосування спеціалізованих комплексів, які мають значну ціну	Можливість заощадити за допомогою застосування загальновживаних апаратних засобів
6.Марочна	Кожна діагностика має бути стандартизованою	Отримання переваги на ринку медійних послуг

Таблиця 5.9. Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Технологічні постачальники	Необхідність пошуку постачальників	Залучення малопопулярних постачальників	Незалежність у прийнятті клієнтських рішень	Надання переваги більш авторитетним компаніям
Висновки:	Незначна	Можливість виходу на ринок є	Постачальники диктують цінову політику на обладнання	Клієнти диктують вимоги до якості	Обмеження існують лише у разі відмови від продукта

Таблиця 5.10. Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Раціональніший ціновий показник	Можливість більш раціонально використати ресурси
2	Надання сервісних послуг	Сервісна підтримка програмної частини

Таблиця 5.11. Порівняльний аналіз сильних та слабких сторін проекту

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні						
			–3	–2	–1	0	+1	+2	+3
1	Раціональніший ціновий показник	13			+				
2	Надання сервісних послуг	15			+				
3	Синхронізованість	20	+						
4	Спектр застосувань	17		+					

Таблиця 5.12. SWOT- аналіз стартап-проекту

Сильні сторони: раціональний ціновий показник, надання сервісних послуг	Слабкі сторони: періодична діагностика, можливості погрішностей при захопленні руху
Можливості: використання для ряду потреб користувачів	Загрози: Незацікавленість клієнтів, втрата авторитету

5.4. Розроблення ринкової стратегії проекту

Таблиця 5.13. Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Медійні та кінокомпанії	Середня	Високий	Середня	Висока
2	Аматорські кіностудіїї та рекламні агенства	Висока	Високий	Середня	Низька

Таблиця 5.14. Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
1	Використання альтернативних технологій та пристроїв	Встановлення нового стандарту якості	Зацікавлення та залучення гігантів у галузі телебачення та кіно	Стратегія диференціації
2	Дешевизна проекту	Раціональніші витрати на обладнання, та послуги	Застосування загальновживаних апаратних рішень замість спеціалізованих комплексів	Стратегія лідерства по витратах

Таблиця 5.15. Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
1	ні	Забирати існуючих та шукати нових	Характеристики апаратної частини	Стратегія виклику лідера

Таблиця 5.16. Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспромо жні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Висока якість послуг	Стратегія диференціації	Синхронізованість	Якість, надійність, точність
2	Мінімальні витрати	Стратегія лідерства по витратах	Широкий спектр застосування	Дешевизна, раціональність, тех. підтримка

5.5. Розроблення маркетингової програми стартап-проекту

Таблиця 5.17. Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Якість	Висока якість, надійність	Надійність
2	Дешевизна	Раціональне використання коштів	Дешевизна

Таблиця 5.18. Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>I. Товар за задумом</td><td colspan="3">Якісні послуги, стандартизована якість послуг та обладнання</td></tr>
<tr><td rowspan="4">II. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>1)Вартість обслуговування,
2)Кількість елементів
3)Строк безвідмовної праці
4)Технологічна собівартість товару</td><td>1) М
2) М
3) М
4) М</td><td>1)Е
2) Пр
3)Нд
4)Тх</td></tr>
<tr><td colspan="3">Якість: міжнародні стандарти якості, високоякісні технології</td></tr>
<tr><td colspan="3">Марка: Кіновиробництво</td></tr>
<tr><td rowspan="2">III. Товар із підкріпленням</td><td colspan="3">До продажу – програмне забезпечення</td></tr>
<tr><td colspan="3">Після продажу – сервісна підтримка</td></tr>
</table>

Таблиця 5.19. Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	500 у.о./од.	200 у. о./од	Високий	Н.50 у.о. – В.180 у.о. (Товар) Н.50 у.о. – В.100 у.о. (Послуга)

Таблиця 5.20. Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Орієнтована на отримання максимальної якості та точності захоплення руху	Поставки якісного, точного та надійного товару	Значна	Договірна система збуту

Таблиця 5.21. Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Зацікавленість в якісному та точному продукті з раціональним використанням ресурсів	Медіа ресурси	Синхронізованість з будь-якими ОС	Зацікавити у покращеннях пов’язаних із зростаючою популярністю послуг	Представлення продукції відправною точкою на шляху до реалістичних інтерфейсів
2	Зацікавленість у великій кількості продукту із дотриманням умов якісності	Медіа ресурси	Широкий спектр застосування	Зацікавити у позитивних сторонах	Представлення якісної роботи з клієнтами

ВИСНОВКИ

У магістерській дисертації запропоновано підхід до створення графічних інтерфейсів. Підхіх не вимагає поглиблених знань OpenGL або WebGl, має можливості інтеграції з сучасними програмами для створення графічних моделей та може бути маштабованим в майбутньому. Також зрозумілий для звичайних користувачів і може мати практичну цінність для навчальних потреб, зокрема, для реалізації кваліфікаційних та атестаційних робіт і проектів студентів кафедри.

В рамках магістерської дисертації було проведено дослідження технологій WebGL та існуючих графічних бібліотек. На основі проведених досліджень отримано наступні результати:

1. Проаналізовано роботу сучасних браузерів. Проведено огляд технологій створення графіки. Приведено базові характеристики та статистику використання у різних країнах світу. Проаналізовано та наведено порівняння векторної та растрової графіки на різних платформах у використанні сучасних браузерах.

2. Розглянуто програми для створення графічних моделей. Описано роботу програми Blender та 3dsMax та наведено базові функції, інтерфейс і особливості використання. Також проаналізовано методи інтеграції з графічними бібліотеками. Наведено порівняння з аналогами, які представлені компаніями розробниками самих програм для створення графічних моделей.

3. Вивчено використання WebGL API, який є базою для графічних бібліотек. За допомогою WebGL також можливе створення графіки в браузерах, але це стає дедалі складніше ніж потрібно для очікуваного результату. Наведено системні вимоги, принцип та структура роботи WebGL додатку. Також наведено порівняння з іншими технологіями.

WebGL є 3D графічною бібліотекою, яка дозволяє сучасним інтернет-браузерам малювати 3D-сцени стандартним і ефективним способом. Рендеринг - це візуалізація процесу створення зображення з моделі за допомогою

комп'ютерної програми. Оскільки цей процес виконується на комп'ютері, існують різні способи отримання таких зображень.

Перша відмінність, використання будь-яких спеціальних графічно-апаратних засобів чи ні. Програмний рендер, використовується в тих випадках, коли всі розрахунки, необхідні для відтворення 3D-сцен виконуються з використанням основного процесора комп'ютера; з іншого боку, використання апаратного рендеринга, є актуальним в тих випадках, коли є графічний процесор (GPU) для обчислення 3D-графіки в реальному часі. З технічної точки зору, апаратний рендеринг є набагато більш ефективним в порівнянні з програмним, бо є спеціалізовані апаратні складові, які обробляють операції. Але, з іншого боку, програмний рендеринг зазвичай більш поширений через брак апаратних залежностей.

Друга відмінність, це процес рендеринга локально або віддалено. Коли зображення, яке має бути відображена складне, то рендеринг, швидше за все, буде відбуватися віддалено. Це випадок 3D-анімаційних фільмів, коли виділені сервери з великою кількістю апаратних ресурсів дозволяють рендерить складні сцени - це серверний рендеринг. Протилежністю цьому є рендеринг, що виконується локально - це клієнтський рендеринг.

WebGL має клієнто-орієнтований підхід; елементи, які складають частини 3D-сцени, зазвичай завантажуються з сервера. Однак, вся подальша обробка, необхідна для отримання зображення виконується локально, за допомогою графічного обладнання клієнта.

Також було проаналізовано два найпопулярніші та конкуруючі бібліотеки, які засновані на базі WebGL. Проведено порівняння у вигляді пошагового створення базових елементів.

4. **З**апропоновано підхід до створення графічних інтерфейсів. Визначено найкращий варіант реалізації в плані швидкості роботи, швидкості створення та загального результату для подальшого застосування. Використання Three js бібліотеки у розробці графічних інтерфейсів має переваги у швидкості та якості розробки на відміну від всіх інших розглянутих технологій. Запронований підхіх

не вимагає поглиблених знань OpenGL або WebGl, має можливості інтеграції з сучасними програмами для створення графічних моделей та може бути маштабованим в майбутньому.

5. Розроблено стартап-проект, який базується на просуванні на ринок розробки графічних інтерфейсів, з використанням запропонованого методу. Проведено дослідження доцільності та рентабельності даного бізнес-проекту та визначено, що комерціалізація проекту є доцільною.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Diego Cantor Brandon Jones. Webgl Beginner's Guide. Become a master of 3D web programming in webgl and javascript, pages 158 – 175, 2012.
2. Kouichi Matsuda Rodger Lea. Webgl Programming Guide: Interactive 3D Graphics Programming with webgl, pages 100 – 117, 2013.
3. Релізація найменування файлів,

URL: https://blog.kongregate.com/tag/game-development/

4. Andy Budd. CSS mastery, pages 212 – 226, 2006.
5. Ilya Grigorik. High Performance Browser Networking, pages 146 – 158, 2013.
6. David Flanagan javascript. Javascript Pocket Reference, pages 54 – 63, 1998.
7. Three.js Optimize Lots of Objects

URL: https://threejsfundamentals.org/threejs/lessons/threejs-optimize-lots-of-objects.html

8. Tony Parisi. Programming 3D Applications with HTML5 and webgl: 3D Animation and Visualization for Web, pages 69 – 85, 2014.
9. Andreas Anyuru. Professional webgl Programming: Developing 3D Graphics for the Web, pages 176 – 185, 2012.
10. Jos Dirksen. Learning Three.js: The javascript 3D Library for webgl, pages 233 – 249, 2013.
11. Isaac Sukin. Game Development with Three.js, pages 312 – 315, 2013.
12. Julien Moreau-Mathis. Babylon.js Essentials, pages 110 – 123, 2016.
13. Julian Chenard. Learning Babylon. Js: Learn the Basics of the 3D Framework Babylon. Js by Creating a Whole Game!, pages 136 – 145, 2017.
14. Eric Meyer. Cascading Style Sheets: The Definitive Guide, pages 202 – 207, 2000.
15. Steve Fulton and Jeff Fulton. HTML5 Canvas, pages 57 – 64, 2011.
16. Billy Lamberta and Keith Peters. Foundation HTML5 Animation with javascript, pages, 2011/
17. Keith J. Grant. CSS in Depth, pages 88 – 103, 2018.

18. Oliver Villar. Learning Blender: A Hands-On Guide to Creating 3D Animated Characters, pages 255 – 261, 2014.
19. Lance Flavell. Beginning Blender: Open Source 3D Modeling, Animation, and Game Design, pages 100 – 115, 2010.
20. Autodesk. 3ds Max 8 Essentials, pages 199 – 223, 2006.
21. Kelly l. Murdock. 3ds Max Bible, pages 147 – 152, 2008.
22. Autodesk. 3ds Max 9 Essentials, pages 28 – 35, 2006.
23. Eric Elliott. Programming javascript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries, pages 67 – 88, 2013.
24. Mark Mayers. A Smarter Way to Learn javascript, pages 95 – 105, 2014.
25. Nicholas C. Zakas. High Performance javascript: Build Faster Web Application Interfaces, pages 355 – 367, 2010.
26. John Resig and Bear Bibeault. Secrets of the javascript Ninja, pages 56 – 62, 2008.
27. Вікіпедія: вільна енциклопедія,
URL: https://uk.wikipedia.org/wiki/Maxwell_Render
28. Вікіпедія: вільна енциклопедія,
URL: https://uk.m.wikipedia.org/wiki/Autodesk_3ds_MAX
29. Міжнародний науковий журнал,
URL: https://www.inter-nauka.com/uploads/public/1505892883187.pdf

ДОДАТОК А

Реферат англійською мовою на тему магістерської дисертації

ABSTRACT

WebGL was originally based on OpenGL ES 2.0 (Embedded Systems), the OpenGL specification for devices such as Apple's iPhone and iPad. But the specification has evolved, become independent, and its main purpose is to ensure portability between different operating systems and devices. The idea of a web interface, real-time rendering has opened up a new universe of possibilities for web-based 3D environments such as video games, scientific and medical imaging. In addition, due to the widespread use of web browsers, these and other 3D applications can be launched on mobile devices such as smartphones and tablets.

WebGL is a 3D graphics library that allows modern web browsers to draw 3D scenes in a standard and efficient way. Rendering is a visualization of the process of creating an image from a model using a computer program. Because this process is performed on a computer, there are various ways to obtain such images.

The first difference is whether or not you use any special graphics hardware. Software rendering is used when all calculations required to play 3D scenes are performed using the main computer processor; on the other hand, the use of hardware rendering is relevant in cases where there is a GPU for calculating real-time 3D graphics. From a technical point of view, hardware rendering is much more efficient than software rendering, as there are specialized hardware components that handle operations. But, on the other hand, software rendering is usually more common because of a lack of hardware dependencies.

The second difference is the process of rendering locally or remotely. When an image that needs to be displayed is complex, the rendering is more likely to occur remotely. This is the case of 3D animated films where dedicated servers with many hardware resources allow for rendering complex scenes - this is server rendering. The opposite is local rendering - client rendering.

WebGL has a customer-centric approach; the elements that make up the 3D scene are usually downloaded from the server. However, all further processing required to obtain the image is performed locally using the client's graphics equipment.

As with any 3D graphics library, WebGL requires the presence of certain components to create 3D scenes.

Items:

- A scene is a 3D space that houses the desired objects: elements (meshes) and light sources that illuminate the elements on either side.
- Camera - point, display the scene space.
- Light source - the element that creates the light. It can be scattered, point, directional
- Mesh is a scene element that consists of geometry and material. 3D object.
- Geometry is a set of vertices that, when generated, are interconnected by graphic primitives.
- Material is a way of displaying and the appearance of an element. This is how he behaves as part of a performance on stage. Shadows and reflections are the simplest examples to demonstrate.
- Texture is an image that can be used within a material to give the appearance of an object.

How WebGL works:

1. The process begins with the creation of an array of vertices. This array contains the attributes of vertices: location in 3D space, texture, color, or normal (light) information. This information is created in JavaScript from 3D model description files (.obj files) or from a library that describes an array of vertices of geometric shapes.
2. The vertices are then sent to the GPU. However, you also need to pass an array of vertex indices to control the transformation of vertices into triangles
3. The GPU reads each vertex from the vertex buffer and drives it through the vertex shaders. The vertex shaders calculate the vertex position on the screen, color, texture coordinates.
4. The GPU connects 3 vertices into triangles using an array of indices.
5. Then there is a rasterization, bringing the image to the format of pixel fragments.
6. The formed pixels pass through the pixel shaders. The pixel shader

calculates the color and depth of each pixel. At the same stage, there is a texture overlay and a lighting calculation. The calculated pixels are placed in the frame buffer (frames).

7. Frame buffer - the last stage of playback. The result of his work is a 2D screen with depth effects.

8. The first argument of this function is the line display, the second is the index in the buffer from which the display will start, drawArrayLength is the number of elements to draw.

JavaScript programming. Working with JavaScript allows you to access and easily access all DOM elements, as opposed to communicating with applets. Since WebGL is programmed in JavaScript, it makes it easier to integrate WebGL applications with other JavaScript libraries such as JQuery and other HTML5 technologies. Automatic memory management: Unlike its sister OpenGL and other technologies, where there are specific operations to manually allocate and free memory, WebGL does not need it. It follows that when JavaScript exits a variable from the scope, the memory it occupies is automatically freed. This greatly simplifies programming, reduces the amount of code, makes it clearer and clearer.

Perseverance: Thanks to modern technological advancements, JavaScript-enabled web browsers are being installed on smartphones and tablets. At the time of writing, the Mozilla Foundation is a program for testing WebGL capabilities on Motorola and Samsung phones. There are similar developments in WebGL support for the Android platform.

Performance: WebGL application performance is comparable to equivalent standalone applications (with some exceptions). This is due to WebGL's ability to access local graphics hardware accelerators. So far, many web technologies for 3D rendering use software rendering.

Zero compilation: Given that WebGL is written in JavaScript, there is no need to pre-compile the code before executing it in your web browser. This allows you to make changes on the fly and see how those changes affect the 3D web application. However, when we touch on the topic of shaders, we will realize that we need some compilation. However, this is done through our graphical hardware, not from our browser.

The library itself is written in JavaScript and is intended for use in the javascript environment. Mostly, this means that it will work on the client side - in a web browser on any device. But with node.js and headless browsers it can also be used on the server side.

Most three.js projects feature real-time 3D graphics where user interaction results in immediate visual feedback. Another type of 3D graphics is either different effects, or artificial characters in movies, or various "renderings" that you can see printed or in a web browser.

The subset of all this is 3D math. 3D graphics cannot be done without mathematics, and computer languages do not understand the default 3D concept. This is where the library comes in, it summarizes those mathematical operations, perhaps optimizes them, and opens up a high-level interface such as Matrix4 or .dot ().

Three.js has its own math library with some classes for 3d math. There are standalone libraries that do this math on their own, but with three, it's just a subset of a much larger system.

You can select an area of three.js that serves as this general 3d world abstraction. A scene graph is a data structure used to describe how objects in a 3D scene (world) relate to each other. In fact, it should not be 3d, as it is a convenient way to describe any vector graphic hierarchy. It is specifically a "tree" made of "nodes" with a "root node" that branches. In three.js, the base class for this data structure is Object3D.

It's almost exactly like a DOM tree. The scene would be similar to <body> and the rest would be industry. At DOM we can position things, but they are quite limited. Rotation usually happens around one axis, and we move things left / right or up / down.

Scene Three is reminiscent of a virtual DOM. We do our operations and set the state in this tree, and when we want to take a visual snapshot of that state (say, in a continuous loop or some user interaction / change of state), we call a renderer (scene). You don't want to update an entire DOM tree when something changes, whereas with the <canvas> element, we have to clear the entire view and then redraw everything, even if only one element has changed position.

An open-source library for creating full-fledged 3D applications and games that work in a web browser without the use of third-party plug-ins and extensions. Babylon JS is close to ThreeJS in its capabilities, but has some built-in features not available in Three JS out of the box. These nice features include the built-in oimo.js physical engine, a simple enough way to create a realistic landscape using a height map. Of course, in three js there are also such opportunities, but they are implemented with the help of various additional applications. However, the functionality of the library has to pay a modest weight of 800 KB.

Babylon.js was created using the TypeScript language. TypeScript compiled and multi-platform language generates pure JavaScript code.